

**NPS ARCHIVE**  
**1969**  
**DIETZLER, A.**

TIME-SHARING TASK CONTROL  
FOR A  
HYBRID COMPUTER SIMULATION LABORATORY

by

Andrew John Dietzler



# United States Naval Postgraduate School



## THESIS

TIME-SHARING TASK CONTROL  
FOR A  
HYBRID COMPUTER SIMULATION LABORATORY

by

Andrew John Dietzler

April 1969

*This document has been approved for public release and sale; its distribution is unlimited.*

LIBRARY

U.S. NAVAL POSTGRADUATE SCHOOL  
MONTEREY, CALIFORNIA

TIME-SHARING TASK CONTROL

FOR A HYBRID COMPUTER

SIMULATION LABORATORY

by

Andrew John Dietzler  
Lieutenant, United States Navy  
B.S.Ch.E., University of Michigan, 1963

Submitted in partial fulfillment of the  
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL  
April 1969

1969

DIETZLER, A.

ABSTRACT

The study of time sharing system parameters and design is undertaken. On-line and hybrid simulation programmer's demands for interactive digital computing time are time inefficient for modern high speed computers, hence the motivation for time shared computing systems. The techniques for achieving time sharing are studied, then applied to the problems of a real time, on-line, hybrid simulation and batch processing system. Subroutines required for implementation of a task oriented time sharing capability are put forward with specific proposals for use. System improvements to accomplish the goal of a general time sharing system are introduced and discussed.

TABLE OF CONTENTS

CHAPTER	PAGE
I. INTRODUCTION	11
II. TIME SHARING TECHNOLOGY	15
Design Considerations	16
System Requirements for Time Sharing	25
III. SYSTEM ENVIRONMENT	31
System Hardware	31
System Software	43
System Relation to a Time Sharing Configuration	50
IV. PACKAGE PHILOSOPHY	52
System Users	52
Queuing	53
Human Factors Applications	58
Engineering Trade-offs	59
Resultant Design	61
V. SOFTWARE DESCRIPTIONS	63
Programming Principles	63
Functional Descriptions	64
Subroutine Discussion and Logic	72
VI. TIME SHARING SYSTEM EMPLOYMENT	97
Program Input, Edit and Debug Mode	97
Step-by-Step Execution and Debugging Mode	97
VII. CONCLUSIONS	101
General Conclusions	101
Recommendations for the System Studied	103
Extensions of the Special Purpose Package	103

## TABLE OF CONTENTS

	PAGE
BIBLIOGRAPHY	105
APPENDIX A CALCULATIONS AND DATA	106
APPENDIX B CODE CORRESPONDENCE LISTING	110
APPENDIX C PACKAGE SUBROUTINE LISTING	112
INITIAL DISTRIBUTION LIST	173
FORM DD 1473	175



## LIST OF FIGURES

FIGURE	PAGE
1 Basic Time Sharing System	26
2 Time Multiplexed A/D/A and Display Channel Linkage	34
3 Memory Division and Channel Access Linkage	35
4 Channel Interlace Registers and I/O Peripheral Configuration	41
5 Graph of Relative CPU Stalling for One TMCC	42
6 Graph of Relative CPU Stalling for Two TMCC's	42
7 Graph of Time vs Words Transferred Under RTM Control	48
8 Graph of Time vs Words Transferred Under High Speed Swapping Driver	49
9 Outline of Proposed System Terminals and Employments	54
10 Scope Formats for Input, Editing and Debug and Step-By-Step Execution	60
11 Functional Arrangement of Input, Editing and Debug Subroutines	65
12 Interrupt Service Routine General Form	67
13 Functional Arrangement of Step-By-Step Execution Subroutines	70
14 Background to Foreground Transfer Calling Sequence	74
15 Foreground to Background Transfer Calling Sequence	77
16 Calling Sequences for Initialization to Enable Step-By-Step Execution	92
17 System Data Flow Under Input, Editing and Debug Mode	98
18 System Data Flow Under Step-By-Step Execution Mode	99



## LIST OF FLOWCHARTS

FLOWCHART	PAGE
1. Subroutine KEYD	73
2. Subroutine EXECUTE	75
3. Subroutine PSK	76
4. Subroutine DRUMRD and DRUMST	78
5. Subroutine DEBUGS	80
6. Subroutine DDEBUG	81
7. Subroutine SPACST	83
8. Subroutine FLASH1 and FLASH2	83
9. Subroutine PEN1ACT and PEN2ACT	85
10. Subroutine KEY1ACT and KEY2ACT	87
11. Subroutine STEPBY	93
12. Subroutine STEP1 and STEP2	95



## TABLE OF SYMBOLS AND ABBREVIATIONS

A/D	Analogue to Digital
ASC II	American Standard Code II
CDC	Control Data Corporation, Minneapolis, Minnesota
CPU	Central Processing Unit
CRT	Cathode Ray Tube
D/A	Digital to Analogue
IBM	International Business Machines Corporation, San Jose, California
I/O	Input or Output
K	Thousand
MAM	Multiple Access to Memory Device
RTM	Real-Time Monitor
SDS	Scientific Data Systems Incorporated, Santa Monica, California
TMCC	Time Multiplexed Communication Channel



## I. INTRODUCTION

The growth of programmer skills and consequent development of additional areas for computer application has brought to the computer a new group of interactive users. The on-line programmer and the hybrid system user have a common digital requirement for short bursts of central processor unit (CPU) activity interspersed with comparatively long intervals of CPU idle time. During the CPU idle intervals the programmer is evaluating results, making decisions and manipulating data in preparation for the next call for a burst of CPU activity. This type of computer use greatly expands a computing system's utility by using the evaluation and reasoning powers of a skilled scientist rather than the limited binary logic capability of a machine. Modern computers have greatly increased speed capability, hence, the CPU idle cycle is becoming increasingly wasteful of sophisticated high speed computing system time. Because the computing system's time is very expensive and the scientist's is relatively cheap, there exists a speed-cost mismatch that if possible should be reversed. It makes more economic sense for the scientist to have additional idle time and the CPU to be continuously active.

The obvious solution to the speed-cost mismatch between modern computing systems and the interactive user is to develop a system environment where several users use one CPU to attain an optimal balance of no CPU idle time and immediate response to the service calls of the users. This optimum expresses the meaning and goal of a time shared computing system.

For purposes of this study time sharing refers to the achieving of effective concurrent execution of independent computational tasks through primarily sequential usage of the components of a single

computer system (as opposed to the technique of providing a computer system for each task). Time sharing systems may be classified on the basis of their service goals into the broad categories:

(1) General Purpose Systems - Here a very minimum of constraint is imposed as to the characteristics of "user" tasks, e.g. his available language, compilers, amounts of memory usage, and timing requirements for program segments. Typically, each of a number of users sees an identical "virtual computer" which appears logically as his own private computer. Design of such systems may emphasize various "quality of service" objectives, e.g. a system may be primarily oriented to provide an average type of service to a very large number of users, or alternatively may stress provision of very high quality service to a quite small number of users.

(2) Special Purpose Systems - In this category much more is pre-specified regarding the tasks to be serviced on a concurrent basis, e.g. program length, memory occupancy, timing (sampling rate) and other data. Large numbers of military and commercial "real-time control" systems fall in this category. Design and implementation of such systems depends largely on a task-oriented approach wherein the characteristics and interrelationships between tasks are documented and an appropriate executive and priority control scheme (either hardware, software, or a combination) implemented to coordinate the various tasks. Within limits, a task may even be defined to include such programming operations as compile, run, debug, thus giving to a special purpose system some of the "user" flexibility of the true general purpose systems.

The objective of this thesis study is to investigate the applicability of time-sharing principles to a particular real-time



environment, that of a digital/analog (hybrid) simulation laboratory.

The system is characterized by:

(1) General Purpose Digital Computer - relatively fast medium scale processing system of conventional (Von Neumann) logical organization.

(2) Analog Computer - medium scale hybrid oriented, featuring flexible digital control modes.

(3) Several user terminals - CRT graphic display and keyboard terminals.

(4) Relatively fast and efficient input/output (I/O) channels oriented primarily to Analog to Digital to Analog (A/D/A) conversion and CRT control.

The system application goals include such tasks as:

(1) Modeling studies involving conventional analog only operation or analog operation with a small amount of digital control.

(2) Iterative mode control of analog computer.

(3) Signal-processing of inputs from on-line transducers.

(4) Experiments involving interaction of digital or analog computer with laboratory bench apparatus.

(5) Presentation of dynamic graphical data on CRT.

(6) Editing and debugging of source programs.

It is known that many instances will occur where a particular task will utilize but a small fraction of the computing system capacity.

Thus as high a degree of concurrency as possible is desired. Particular questions to which this study is addressed include:

(1) Definition of anticipated processing environment on a task-oriented basis.

(2) Typical timings and loadings.

(3) Method of priority and interrupt control.

(4) Determination of "free programming" capability feasible for implementation in "spare time" of system.

(5) Memory allocation schemes and their effect on timing and efficiency.

(6) Special limitations of the system.

(7) Architecture of proposed operating system.

The study of time sharing systems will be accomplished by progressing from the study of general purpose time sharing principles to the more specific problems facing the system designer in the implementation of a special purpose system.

The study of time sharing systems first from the view point of the system designer's general non-system oriented problems provides a good building block for later design work. Adhering to the principle that "to learn is to do" the design and implementation of a special purpose, task oriented time sharing capability was undertaken as a study vehicle. The design and implementation phase of this study was centered on a real-time hybrid computer simulation laboratory.

Some general conclusions about time sharing system hardware and software were filtered out of the specific study undertaken. From these, specific recommendations for the system studied can be made regarding implementation of an efficient time sharing capability.

## II. TIME SHARING TECHNOLOGY

The idea of one central computing processor serving a group of user's requests has been discussed at length by several authors.<sup>1, 2, 3.</sup> The actual implementation of such systems is tailored to fit the particular environment of each system. The software designer has encountered two separate environments requiring him to perform one of two tasks. First, more historically now, the task has been one of upgrading existing systems by writing significant software supervisors to do the book-keeping and core management associated with multiple user systems. Experience proved these systems slow but tolerable for a very few users and unacceptable for any practical number of terminals. A transition was made when the additional requirements were specified for hardware designers to implement as improvements into new computing systems. These hardware improvements upgraded the efficiency of the overall system by a reduction in required software supervisor size and increase in the speed of service to an increased number of terminals, providing a class of computing systems easily adapted to a time sharing use. The second environment for software designers to work in has become the basis of present software design of time sharing systems.

1

M.V. Wilkes, "The Design of Multiple Access Computer Systems", The Computer Journal, V. 10, N. 1, P. 1-9, May 1967

2

M.V. Wilkes and R.M. Needham, "The Design of Multiple Access Computer Systems: Part 2", The Computer Journal, V. 10, N. 4, P. 315-320, February 1968.

3

C.G. Bell, "Fundamentals of Time Shared Computers", Computer Design, V. 7, P. 44-47+, February 1968, and V. 8, P. 28-43, March 1968.

## A. DESIGN CONSIDERATIONS

In the design of time sharing systems, consideration must be given to some essential problems where software choices are available. Trade-offs between core economy and speed efficiency, between general unrestricted use and supervisor complexity become determining factors in eventual selection of system design. These trade-offs arise from consideration of problems in the areas of memory management, execution or servicing priority, inter-program isolation, input and output communications and their media, and human factors engineering.

### 1. Memory Management

Memory Management herein refers to high speed core memory and the respectively slower magnetic drum, disc, and tape bulk memory devices. The following is a discussion of memory management schemes progressing from the simplest used to the most complex state of the art procedures. None of these schemes can be chosen over its companions as optimal without study of the system environment it must work in.

The simplest memory management method employable in a time sharing system is a swapping procedure which swaps out of core one program at the end of its execution cycle and swaps into core the next program for its allotted execution cycle. This swapping method allows in core only the system supervisor and one active program at any one time. All other programs are saved on the slower memory devices. This procedure has the advantage that if there is a significant error in the executing program and a system failure occurs, only the offending program is lost.

A more adaptive scheme has been developed in which the supervisor remains in residence and several user areas are allocated from the remaining core. If any object programs are larger than their



respective allocated core area, the additional storage requirement is met with bulk memory devices such as drums or discs. This implies that at any one time core will have resident the supervisor and a number of unexecuted or partially executed programs or parts of programs. The portion of a program not in core will be stored on the slower bulk memory device ready for calling when needed in core. When a partial program in core requires a portion of the program stored externally during execution, its execution terminates and the required portion is transferred into core to satisfy the reference.

This second scheme relies heavily on an interesting observed phenomenon characteristic of most object programs. Programs have only a small section of the whole program active at any one time, most references being to locations in the near vicinity of the instruction in the instruction register. Thus, there is a large portion of memory being used to hold resident program information that is not being accessed or used to control the the computations in progress. This portion of core that is being uneconomically used as bulk storage is made available for additional programs using a swapping procedure where low useage frequency locations are stored in the bulk memory areas, rather than resident in core. The active portion of the program consequently stays in this memory window by application of chaining techniques. The completed portions of the object program are paged into bulk memory and replaced by unexecuted portions of the object program. This technique, known as paging, is used to segment programs into convenient blocks for bursts of execution activity and bursts of transfer activity between bulk memory and core. Paging allows execution of programs much larger than the actual unshared size of the system core memory.

There are many variations derived from the two basic ideas of swapping and paging. One is a segmentation of core and bulk memory into blocks useable for one terminal only. Paging is then used to access areas of interest. This can be made less wasteful of core by making the segments dynamic in size. Time used for doing the bookkeeping required to trace several dynamic boundaries, and time consumed in paging are all added to the program execution time. The trade-off here is between which is more expensive, additional core or execution time, and that depends entirely on the system use.

Two other attempts to compromise the cost of additional core have been based on special purpose core memory. One is read only memory and is used for table look-up of tabulated function values or storage of code translation tables. This saves time because these tables are always resident and are immediately accessible. Normally tables of this type are kept on bulk memory and paging is required to access them. Non-executable memory is another name for core that is read and write only. Dynamic lists and temporary storage locations are held in this area of core. Non-executable memory is not directly addressable by the program counter, or location register, but is directly addressable by the operand register. Executable instructions held in this type of core are transferred into executable core prior to execution. This is a high speed swapping scheme but still takes time and requires supervisory activity. The trade-off is using both of these special purpose core applications is in favor of a high speed requirement versus a low cost. Systems with this type of high speed bulk memory are applied to problems where minimum user time and high execution speed are required.

## 2. Execution Priority

Execution schemes all have a common requirement for an instruction register separate from core. This means that any instruction to be executed must normally be transferred from memory to an instruction register. There is no requirement on the number of registers in any computing system other than cost and complexity of the supervisor. Most systems have one instruction register, thus can execute only one program or ordered sequence of instructions at a time. Which program will be executed first or last is a question of which program has highest priority.

A hardware assignment of priority, a software assignment of priority based on internal parameters such as accumulated user time and memory requirement, or assignment of priority based on user originated verbal or written rules extended to the machine are the three basic methods used to determine execution priority. Most systems are a combination of the three, each on a different level separated by a classification of the type of user to which each method applies.

Hardware assignment of priority assumes a computing system designed for more than one user and gives priority to execution requests on the basis of where they originate. The request for execution is serviced under a software supervisor that could be designed to interrupt CPU activity to service high priority entries. The supervisor could instead establish a position in a timing queue, the timing position a function of the priority's level in the system hierarchy combined with other parameters. The disadvantage in this method is that user requirements differ and to get the optimal priority-job match each user must seek out the terminal that can provide the priority level required for his jobs rather than

use the terminal most convenient to his physical location. The advantage in such a system is that each terminal could be designed for a special purpose and have additional hardware available to increase the power of the terminal. This is common at real-time hybrid simulation terminals where additional linkage to the digital computer is available for the special purpose user.

Software assignment of priority is done on the basis of entering control arguments or with timing schemes. The priority could be determined by a timing consideration, giving each program a specific time slice for execution. Programs not completed can be saved by swapping or updated by paging, depending on the memory allocation scheme. The time slice size can be equal portions for each program, a "round robin" method, or a time slice could be lengthened as the number of re-starts increases for a particular program, or a simple run until completed first-in first-out queue could be used. This last method is in general not acceptable because one user can tie up the system for an intolerable length of time. The more favored method is to allow a maximum time limit for each program. When the time limit is exceeded the long program is interrupted and the supervisor initiates execution of the next program in the queue. Another modification on this is to interrupt at the time limit or when input/output (I/O) is initiated. Because the active program normally must wait for I/O to complete before execution can continue it is interrupted. The I/O is run with interlace and multiplexed channels during the execution of the next program up in the queue.

The simplest and most versatile priority assignment possible is a verbal agreement of priority that system users determine among themselves. This method is restricted to small systems with priority



reassignable terminals and conversant people. Timing considerations should be input to the software supervisor based on a set of commonly agreed on parameters modified to best serve the needs of the system users. A system architecture of this type is uncommon, but, when taken proper advantage of, will yield the best results.

Modern time sharing systems employ a mix of these priority establishment methods. A verbal agreement is reached on the execution priorities available and what priority class each request falls into. The priority assignment could be based on a listing of servicing parameters. Certain classes of requests could be tied to a hardware priority hierarchy and given service based on a software evaluation of where it falls in the combined priorities.

### 3. Isolation

Program isolation is required if the computing system is to avoid loss of temporarily inactive programs that are stored, but waiting for a higher priority program to finish in the queue. The method used to accomplish isolation in most systems is very dependent on and normally incorporated in the memory allocation scheme.

An isolation scheme used that is independent of the memory allocation scheme uses what is known as programmable lockouts. Programmable writing lockouts, when installed, can be set by the supervisor to allow writing only in the area reserved for the presently active routine. After execution the supervisor resets the lockout, the unlocks the area corresponding to the next program in the execution queue. By using appropriate chaining and mutual protection complete dynamic isolation can be maintained.

Systems with a paging memory allocation scheme accomplish isolation by incorporating a page table that indicates which segments

of memory may be accessed by each program. The inclusion of the supervisor in this table effectively protects the system and other users. The page table is a dynamic table, changing at the end of each executed program to indicate what areas are available for the next program in the queue and what areas are reserved for each program in the system. The option of saving a program area for re-execution after debugging is sometimes included in the supervisor, in which case the page table will retain a program after execution until the program is specifically cleared.

#### 4. System Communication

Time shared computing systems have two levels of communication, internal and external. External communications link the machine to its users. This is a two-way linkage established at the user terminals. The terminals require a storage capability to buffer the millisecond speed of the machine I/O down to the time mode of the system user. In time shared systems, the input terminals are normally used for output, if they are remote terminals. Most systems include an option of several methods for feedback of results, microfiche, magnetic tape, binary paper tapes, line printer output, and graphical display on a CRT. The options depend on the complexity of the remote terminals and type of communication links used. The basic requirement is for some form of immediate response or feedback to maintain communications between the input terminal user and the CPU. This consists of some indication of what has been accomplished at the CPU, what is occurring, and what will occur next.

The internal communication system links memory, the central processing unit, and I/O peripherals. The internal communication requirement for time shared systems is extreme economy to insure that

system hardware spends very little time waiting for the command signals required for continued I/O activity. Hence, the internal communication system should employ a maximum amount of independent and parallel activity. Ideally, internal channels are at least the same width as the machine word for parallel transmissions. These channels should be time multiplexed on a demand basis and use independent registers to allow each channel to independently keep track of its I/O listings.

#### 5. Human Factors Engineering

Time sharing systems are uniquely adapted to on-line programming and to be of optimal use should be designed to the specifications of the human who is providing the input data and evaluating the computational results as feedback on his activity. This study indicates what some of the more important parameters are and in what range of values they tend to optimize the man-machine interface. Two general factors important to this study are the minimization of confusion for system users and the optimal system response times to provide attractive time sharing service.

To minimize (hopefully eliminate) the confusion of terminal users all functions should be labeled clearly. The labels can be augmented by comments on the output media that will lead the user to the next proper step in the execution of his particular request. Whenever possible activity that is common to several media should be presented with an identical set of rules and format for each media. All labels should be clear and unambiguous. Factors such as character size and intensity on CRT's, line spacing on media of the terminal, format of feedback (underlined, boxed, offset, etc.) and distance from the user (should be less than 29 inches) should be optimized for each application. Errors should be easily

correctable and if possible self-identifying.

Basic studies<sup>4</sup> indicate that human physical movement rates are no greater than ten operations per second and visual evaluation rates slow to a speed of two per second. Reading rates normally range from two to eight words per second but for persons checking program feedback the higher reading rates are optimistic as was indicated above. Normally, remote computer input terminals use a teletype keyboard. Typing rates vary significantly, but an upper limit of about seven characters per second would rarely be reached and input rates would be anticipated at a rate closer to two characters per second. These functional rates help establish an allowable maximum time for I/O functions at the remote terminal. The keyboard input processor must be faster than a tenth of a second, a physical upper limit, to allow input typing.

Studies made by human factors engineers<sup>5,6</sup> indicate that time sharing systems should have two basic rates of servicing. The on-line programmer is best served by a conversational rate or fast response of less than one second. However, it has been found<sup>5</sup> that

4

Chapanis, Alphonse, Man-Machine Engineering, Wadsworth Publishing Company, Inc. 1965.

5

Nickerson, Raymond S., Elkind, Jerome I., and Carbonell, James R., "Human Factors and the Design of Time Sharing Computer systems", Human Factors, V. 10, N. 2, P. 127-133, April 1968

6

Nickerson, Raymond S., Elkind, Jerome I., and Carbonell, James R., "On the Psychological Importance of Time in a Time Sharing System", Human Factors, V. 10, N. 2, P. 135-142, April 1968.



waits of up to ten seconds are tolerable if the user knows that the computer will deliver a reasonable and useful result at the end of this wait. If service cannot be provided by the system at this rate, a very slow rate of ten to sixty minutes becomes acceptable. The latter slower rate makes it possible for the user to perform secondary tasks that allow the user to "time share" his efforts between computer interaction and secondary tasks. Another important concept must be adhered to in this dual servicing rate system, that of consistency. The response time must be predictable to avoid loss in continuity of the user's activity. Use of response time indicators would be ideal for the programmer.

#### B. SYSTEM REQUIREMENTS FOR TIME SHARING

Implementation of a time shared system design requires specification of the prospective environment of the system. Once the environment has been determined, the minimum required hardware for reasonable system capability can be proposed. Basic time shared system applications and configurations will be discussed. Progressing from more simple systems an analysis of desirable system capabilities will lead to discussion of more complex time sharing systems.

A system environment of several terminals, each requesting batch-type processing of jobs, table look-ups, and in general non-interactive off-line requests could require a design based only on equal time allocations for all users. The least complex system for this application is a medium sized core memory, rapid auxiliary memory such as magnetic drum or disc, more than one teletype user terminal and an interlaced series of time multiplexed communication channels. Figure 1 schematically represents this type of system. The remote terminals are used for program input and after completion of run time segments, a display of the interim and final results. Single terminal to CPU communication

# BASIC TIME SHARING SYSTEM

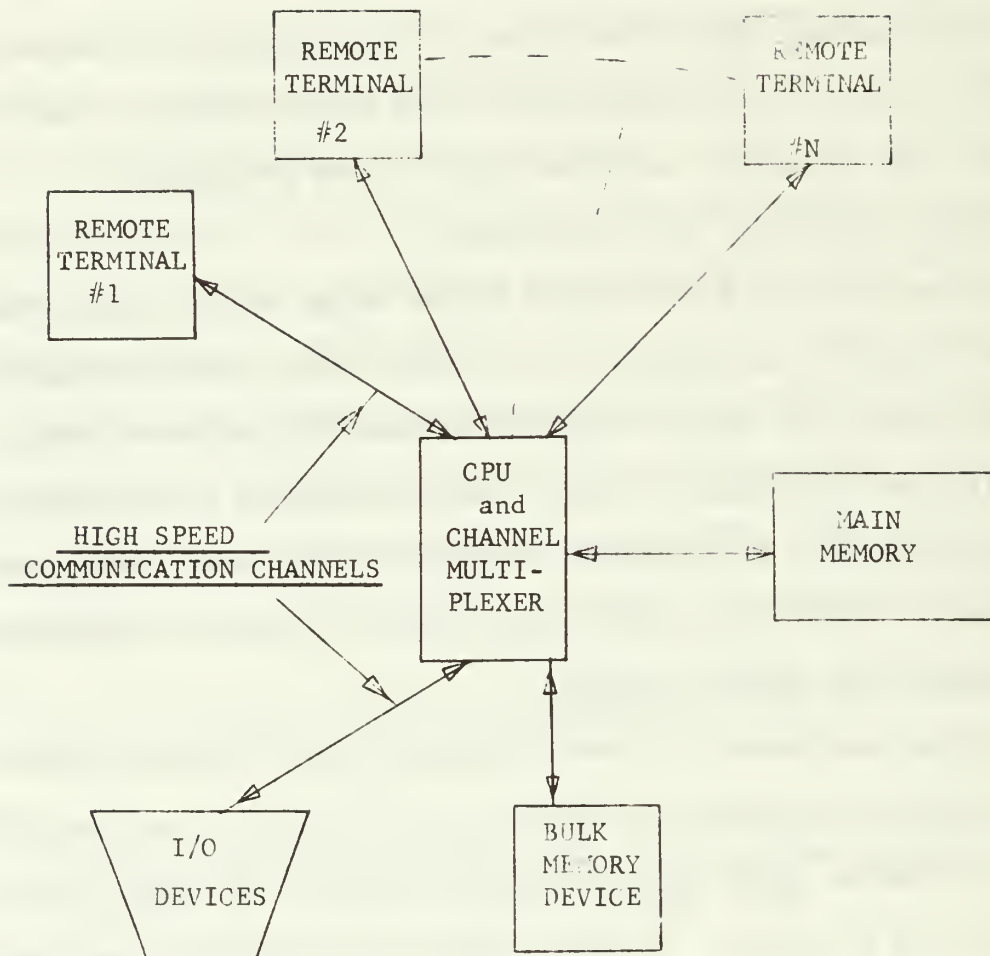


FIGURE 1

channels are required. The system is never physically idle because it is always either serving a user's request for processing or storing additional input, both activities within the supervising software program. Program input done at the remote terminals is stored by the CPU on magnetic discs or drums (bulk memory) until run time. At completion of execution the results are put back on the I/O channel and recorded at the remote terminal for user evaluation. Without time multiplexed access to memory and a channel interlace the CPU performs each of the above steps serially. To improve the system response time the communication channels are time multiplexed with the CPU for access to main memory and each user terminal's I/O buffers.

The use of interlace registers on each time multiplexed channel for control of I/O data lists and programs allows execution of programs concurrent with I/O operations. This is normally capitalized on by starting the I/O for a presently active program then initiating execution of the next program in the queue. The multiplexing allows memory access for the active peripheral I/O device between the memory accesses required by the CPU for execution of a program. The high degree of parallelism in this activity helps meet the speed specifications of time sharing systems. There is a trade-off here between increased I/O speed and decreased CPU speed.

The time multiplexing slows the CPU execution rate by only a very small amount in most systems. The transfer speed capability of the I/O device is directly proportional to the amount of slow down in the CPU execution thus the CPU is "stalled" whenever I/O rates approach the execution rates of the CPU. The fastest I/O devices in common usage are still in the ratio of 1:1000 to the execution rates of common CPU's, thus complete stalling is achieved when 1000 fast I/O

devices are operating simultaneously. This is not likely to occur because most time sharing systems service far fewer than 1000 terminals. When complete stalling does occur the system has reverted back to the inefficiency of system hold-up during I/O, thus the multiplexing and interlace features are nullified.

For larger systems, inclusion of a small independent memory at the user terminals for buffer control and buffering between the I/O devices, bulk, and main memory eliminates I/O overload of CPU execution and memory access completely. This improvement again expands the capability of a system to service many users at even higher speeds. More advanced systems now use a multiprocessor environment <sup>7</sup> to increase the speed efficiency and user capability. These processor areas are normally small, contain the I/O lists and chaining pointers, and are becoming a good application area for thin film memory. The additional core required to meet the higher speed specification increases the cost and complexity of the system.

A slightly different approach to the multiple processor solution of the stalling problem is to interface each individual terminal to memory through a coupler that will provide I/O device to memory access in a first-in first-out queue. The multiple door to memory coupler still has a potential overload problem similar to that of a multiplexer. The overload could occur if the multiple door entry area in core coincides with the CPU entry area in core. Software separation of these areas will eliminate the problem and the multiple door then is of nearly the same convenience as the individual buffer. The reduced costs gained by eliminating the independent user terminal memory, is traded for a slight

7

Kuck, David J., "Illiac IV Software and Application Programming", IEEE Transactions on Computers, Vol. C-17, No. 8, August 1968.



loss in CPU efficiency and less available core for active program users.

Time sharing systems that are used for scientific applications and batch processing have an additional requirement that calls for user priority assignment that is less rigid than in the simple system of Figure 1. Some scientific applications such as hybrid calculations and real-time simulations cannot be arbitrarily terminated at the end of a specified time interval, as in the simple system, without the chance for loss of significant data, if not complete disruption of an experiment. These problems require service for variable lengths of time on a run until completed basis.

This implies a need for an interrupt system that can be activated on call and will discriminate in favor of some privileged users who have special requirements. The interrupt systems in use, in order of increasing complexity and versatility, are equal priority with software logic, a hardware designer-assigned priority interrupt system and a priority interrupt system with programmable priorities which allow users or the system supervisor to assign a priority level based on changing environmental situations. The utility for reassignable priority interrupts allows the reassignment of a terminal from use as a real time terminal to a batch processor with no excessive hardware or software requirement.

A major consideration in systems that employ a time sharing capability is a memory lockout ability that allows only the active user's allocation in core to be changed by writing. The protection feature must carry over to the slower area of disc or drum to prevent loss of any portion of a user program, or the supervisor. The memory lockouts normally employed are a manual lockout and a programmable lockout feature. Both techniques require a hardware signal interception

scheme to inhibit writing in a section of memory. Other schemes are available such as re-interpretation of addresses sent in at each terminal so that they address a different section of memory, divided between core and a virtual bulk memory. This requires either an input terminal independent core for a preprocessor, or a discrimination procedure at the remote terminal-CPU interface.

The initial time sharing system example had a limited capability. A further specification for more terminals increases the speed required for execution, supervisory activity, and I/O. Speed capability is attained by use of additional registers (interlace) and core (independent terminal memory) to attain parallel CPU execution and I/O. To supervise this activity the supervisory software is more complex. Because supervisory software internal time requirements can become excessive, additional hardware such as memory lockouts, interrupts, and chaining registers become necessary. Specification that the system is to have remote terminals in different types of environment means that priority interrupts are necessary to help prevent software overloading. Because there is a limit to how much core memory is allowable, high speed bulk memory devices connected to memory by high speed, interlaced, time multiplexed channels are required. The implementation of the suggested system improvements has several forms, all linked to the environment of the computing system to optimize the solutions to the problems discussed under system technology.

### III. SYSTEM ENVIRONMENT

The study of an existing non-time shared system environment was undertaken to determine the applicability of the above time shared system design considerations. Software requirements are dictated by the hardware efficiencies and deficiencies encountered in a system, hence a thorough understanding of a system's hardware capabilities is required in the initial phase of time sharing design. Analysis of the system software to determine how it fits into a time sharing application is then undertaken. The standing system to be studied is a hybrid computing facility with one analog computing station, two CRT display consoles and one digital control console.

#### A. SYSTEM HARDWARE

A brief description of the physical components of the system and how each is linked together follows. Particular attention to time sharing requirements and system capabilities will yield an estimation of how well the system hardware is adapted to time sharing applications. In this study hardware deficiencies will become obvious and ways to circumvent these difficulties are discussed.

##### 1. Analog Computer

The analog station has the standard analog system components of operational amplifiers, potentiometers, precision resistors and capacitors. In addition, there is a precision clock, digital counters and an extensive logical patching network for control of analog and digital computations, control of problem logic for extensive system simulation, and data reduction using the system's sampling capabilities and fast fourier transform analysis. This analog system is capable of operation under digital control or capable of taking

control of the digital system on an interrupt basis. Normal batch processing (background operation) can be momentarily interrupted for analog servicing, then control returned to background as soon as the analog data requirements are satisfied.

Normal analog requirements for digital service call for input of new settings for potentiometers, new initial condition settings, re-setting of differential analyzer thresholds, sampling of selected output values, data storage, processing and I/O. Digital services are initiated by two types of interrupts, analog logic interrupts and data channel interrupts.

Analog logic interrupts are tied to timing counters or differential analyzer outputs through the logic patchboard. When triggered, an interrupt shifts the digital system from its background activity to a service routine written by the user to perform the above mentioned types of function applicable to his analog problem. Upon completion, control is returned to the interrupted background process. A second set of interrupts are linked to the digital - analog interface and are triggered automatically whenever an interface buffer transfer, digital to analog (D/A), analog to digital (A/D), or the analog component address and voltage buffer, has been completed. The interrupt will initiate any end of transfer activity the user desires.

The transfer of data on the D/A, A/D, and control (analog component address and voltage reading), channels is done through a second port to main core memory. The multiple access to memory (MAM) device allows the analog computer to access a block of memory for data buffering and leave unaffected background processing until the two activities address the same block of core. When simultaneous access is required into the same block of core, the MAM



device has priority and accesses first. The MAM multiplexes requests for memory access between (see Figure 2) the A/D, D/A, and control channels of the analog computer and the display buffers being read by the display hardware to drive the CRT amplifier.

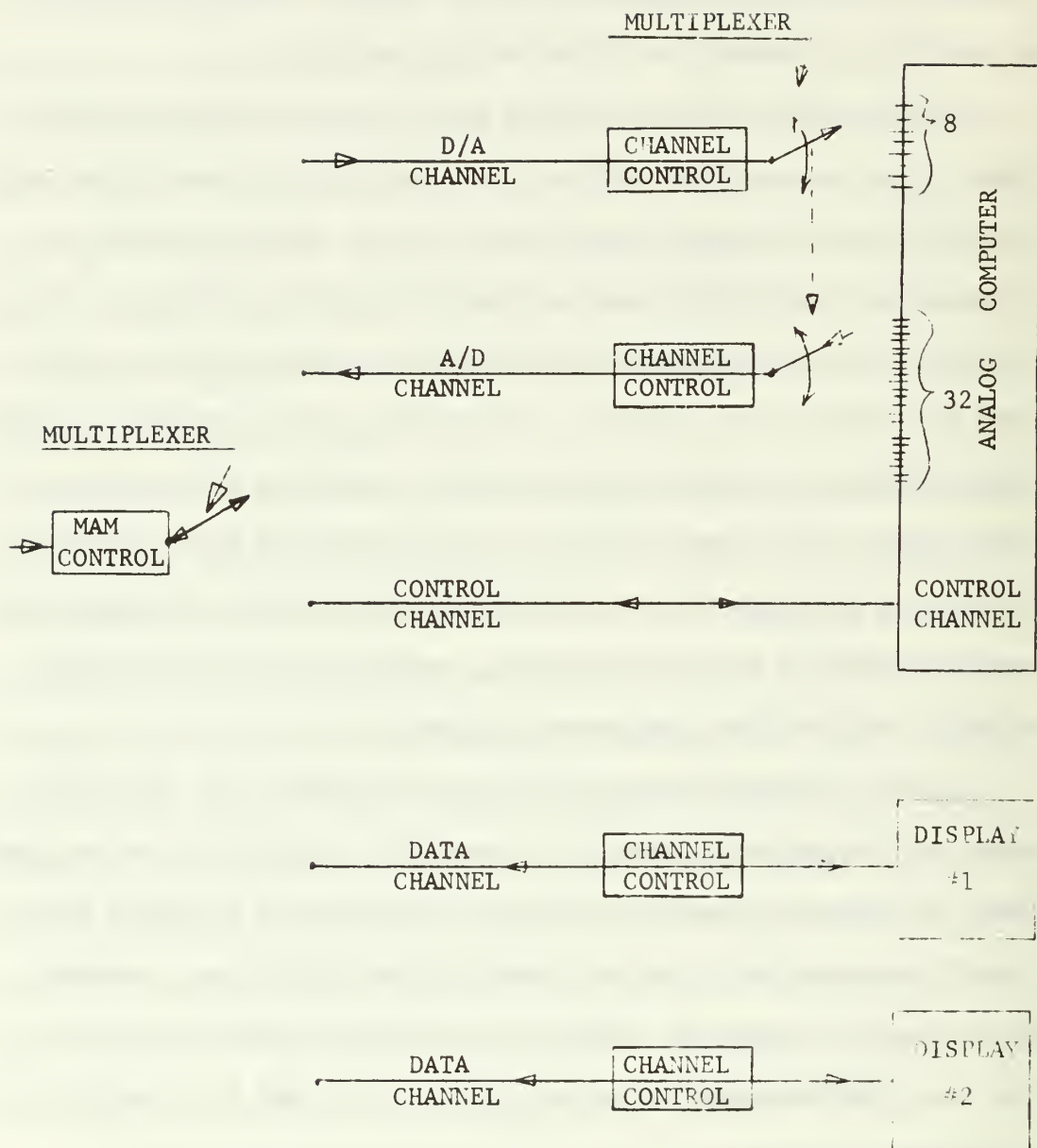
The interrupt system connecting the CPU and the analog computer is ideal for a time sharing application. The logically controlled and end of I/O activity interrupts allow calls for digital service to enter and exit the batch queue without a disruptive influence, once the called routines have been assembled and compiled into an octal form and stored in the system. The channels are multiplexed and have access to memory through a multiple door, providing I/O buffering at high speeds with minimal stalling of CPU controlled batch processes. The D/A and A/D channels are effectively interlaced so that each can run independently and simultaneously, again ideal for the parallel activity required for time shared systems.

Figure 3 indicates that core memory is divided into three separate and independent blocks of a lower 8K\*, a middle 8K and an upper 16K. The system supervisor resides in the lower 8K of memory with a small dynamic list of tables stored at the top of core referred to as upper in Figure 3. The analog and display buffers must be in the upper 24K of memory to be accessible to the MAM for transfer to the respective device.

## 2. Display Units

Each display console has a twenty-three inch diagonal CRT, a light pen, and a teletypewriter keyboard. The scope CRT is coated with a fast time constant non-memory type phosphor, and has beam

\*All references to list lengths and memory block sizes are in thousands (K) and decimal radix. e.g.  $8K = 8000_{10} = 017500_8$



TIME MULTIPLEXED A/D/A and  
DISPLAY CHANNEL LINKAGE

FIGURE 2

MEMORY DIVISION and CHANNEL ACCESS LINKAGE

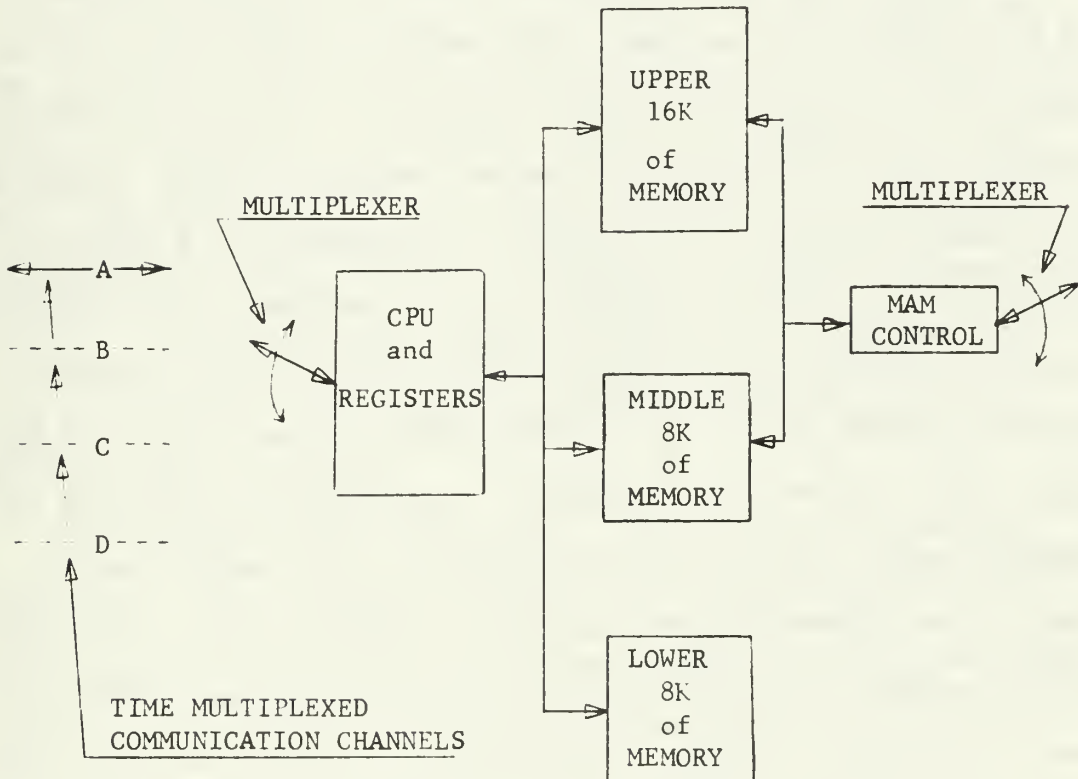


FIGURE 3

writing speeds from 7.5 to 18 microseconds for characters and maximum vector writing time of 52 microseconds on a useable display area 12-1/2 inches horizontally and 13-1/2 inches vertically. The flicker free refresh rate is 45 frames per second. The displays communicate with main memory through the second port to memory (MAM) and are time multiplexed with service calls for A/D and D/A buffer activity.

The displays use software defined buffers in core for storage of the data presented on the scope. Each word is accessed from core and transferred to a single word buffer for display. The word is read and printed in one of three modes: character, vector, or point. Depending on the mode specified, the single word is interpreted as a four character word of six bits per character, an x-y position for the beginning or end of a vector, or as a single point to be displayed. The two controlling words specify the starting address and word count for the display list, and the mode of display. These words are stored in independent control registers for each display, which enables the displays to run independently of each other. Whenever a word count and starting address is specified as zero, the end of display list interrupt is initiated. Activity is suspended until either the list is restarted with the word count and starting address of another display buffer, or end display action is completed.

The light pen is a light sensing diode that suspends display activity within 1.75 microseconds after sensing the writing beam and triggers an interrupt. The address of the word in the display buffer being written by the writing beam at the time of the light pen strike is available to the digital interface and input with software. In the case of vector or point modes, only the buffer word address of the strike is available, however in character mode the character



position in the buffer word is also available for processing.

The teletype keyboard used at each display console is a system-standard keyboard for inputting characters\* with additional keys that provide convenient functions for scope presentation. Accompanying the keyboard is a function panel that has an additional 32 keys (numbered 0 - 31) for specific programmed functions if desired. A key stroke on either the teletype keyboard or the function panel triggers the same interrupt and provides the digital interface with an octal code indicating which key was depressed. Software is required to input this coded data for processing. The keyboard code is the American Standard Code II (ASC II) which the display hardware is designed to display. The function panel code is a sequential octal numbering that corresponds to the struck key, which is convenient for software connection techniques.

The interrupt system installed at these displays is ideally suited for a time sharing application. Entry may be made into the batch queue and service routines called without disrupting the batch programs in the system. This presumes some software provision for saving and restoring is made in the interrupt software. These consoles could be used for input of symbolic programs and processing of them in the batch queue in an orderly fashion. The multiplexed MAM and effectively interlaced channels provide a high speed parallelism required for time sharing applications. Because of the MAM, access to the drum is efficiently initiated and interlaced with other CPU activity. A requirement for at least two channels for the

\*See Appendix B for listing of keys and corresponding codes.

CPU becomes apparent at this point. For maximum parallel activity program input to the drum could progress on one channel, multiplexed with I/O activity from the batch queue on a second channel and with the hybrid I/O on a third channel.

### 3. Main Control Console

The control console has a teletypewriter for input and an octal display of the accumulator (A), extended accumulator (B), three index (X1, X2 and X3), flag (F), program counter (P), and instruction (I) registers with indicators for active channel and peripheral unit. The control console is the man-machine linking interface where system activity is presented visually for operator evaluation. An on-line programmer may take control of the system for input and execution of instructions in a step by step mode at this console by manipulation of the step, idle, run, and hold control switches. Using the teletypewriter, program execution may be initiated or terminated. Two programmable interrupts and six sense switches are available for hand operation and the on-line programmer may utilize these to perform specialized functions during execution of his program.

### 4. Interrupt System

The priority interrupt system is an absolute priority hierarchy which cannot be changed without hardware modification. The triggering of an interrupt causes the system to execute the instruction in the core location corresponding to the interrupt's priority, hence, interrupt 027\* has a priority in the interrupt hierarchy of 027 from the highest and executes the instruction in location 027. This implies that there are 026 interrupts that can interrupt

\*Numbers preceded by a zero are octal i.e. 027 is read "octal twenty-seven".

when interrupt 027 has occurred and has not been cleared. The highest priority interrupt triggered always takes precedence over a lower priority interrupt.

A priority interrupt has three states: inactive, waiting, and active. The inactive state is self-descriptive, the interrupt will not react to, and does not have any triggering pulse on it. Hence, it is in a do-nothing state. Waiting state is reached by arming the interrupt with a software instruction so that it can respond to a triggering pulse that would initiate the execution of the interrupt's corresponding instruction. The active state is reached if two criteria are met. First, the interrupt must have been armed, that is, in the waiting state. Second, there must be no higher priority interrupts active at the time of the triggering pulse. If a higher priority interrupt is active, the triggering pulse is saved and after the presently active high priority interrupt is cleared the triggering pulse is then acted upon to put the waiting lower priority interrupt active. The only automatic hardware functions are the trigger pulse, priority check, the save of the trigger pulse if a higher priority interrupt is active and uncleared, and the branch to the priority specified location upon going active. This means that software interrupt servicing routines that can interrupt any other routine must provide return to the interrupted routine. Additionally, software interrupt servicing routines must not clear a newly active interrupt until the newly active routine can be interrupted by a lower priority interrupt without loss of continuity. When an active interrupt is cleared, the only branch that can automatically occur is on a saved interrupt waiting to go active from the waiting state. Lower priority routines, once active

and then interrupted, are not automatically returned to upon the clearing of a higher priority interrupt. This responsibility is the software programmer's.

The priority interrupt system provides the capability for temporarily suspending execution of one program in favor of execution of a second. The priority interrupt system is excellently adapted for a time sharing environment and eliminates considerable amounts of software with its built in logic for waiting states. Because each interrupt may be connected to any routine with ease, the interrupt user may be varied according to changing system configuration.

#### 5. Time Multiplexed Communication Channels and Peripheral Devices

The system peripherals are linked to main memory by high data rate time multiplexed communication channels (TMCC) each with a set of registers that allow for interlaced activity between the CPU and the channels. The I/O channels operate at the speed of the I/O peripheral attached. Figure 4 illustrates the configuration of the channel with its attached peripherals and interlacing registers.

The channel demands for memory access are considerably slower than the CPU in this system. Hence, individual peripheral memory buffers are not necessary. Figure 5 indicates the relative percentage of CPU reduction in execution speed for interlaced I/O for the system device. Each channel may have I/O on only one device at a time and the TMCC control unit can service up to four channels, although only one channel is installed. The TMCC Control Unit connects to core memory through the CPU and services calls for memory access from the channels when they arise.

This system has a high speed, 800 lines per minute, 132



# CHANNEL INTERLACE REGISTERS and PERIPHERAL CONFIGURATION

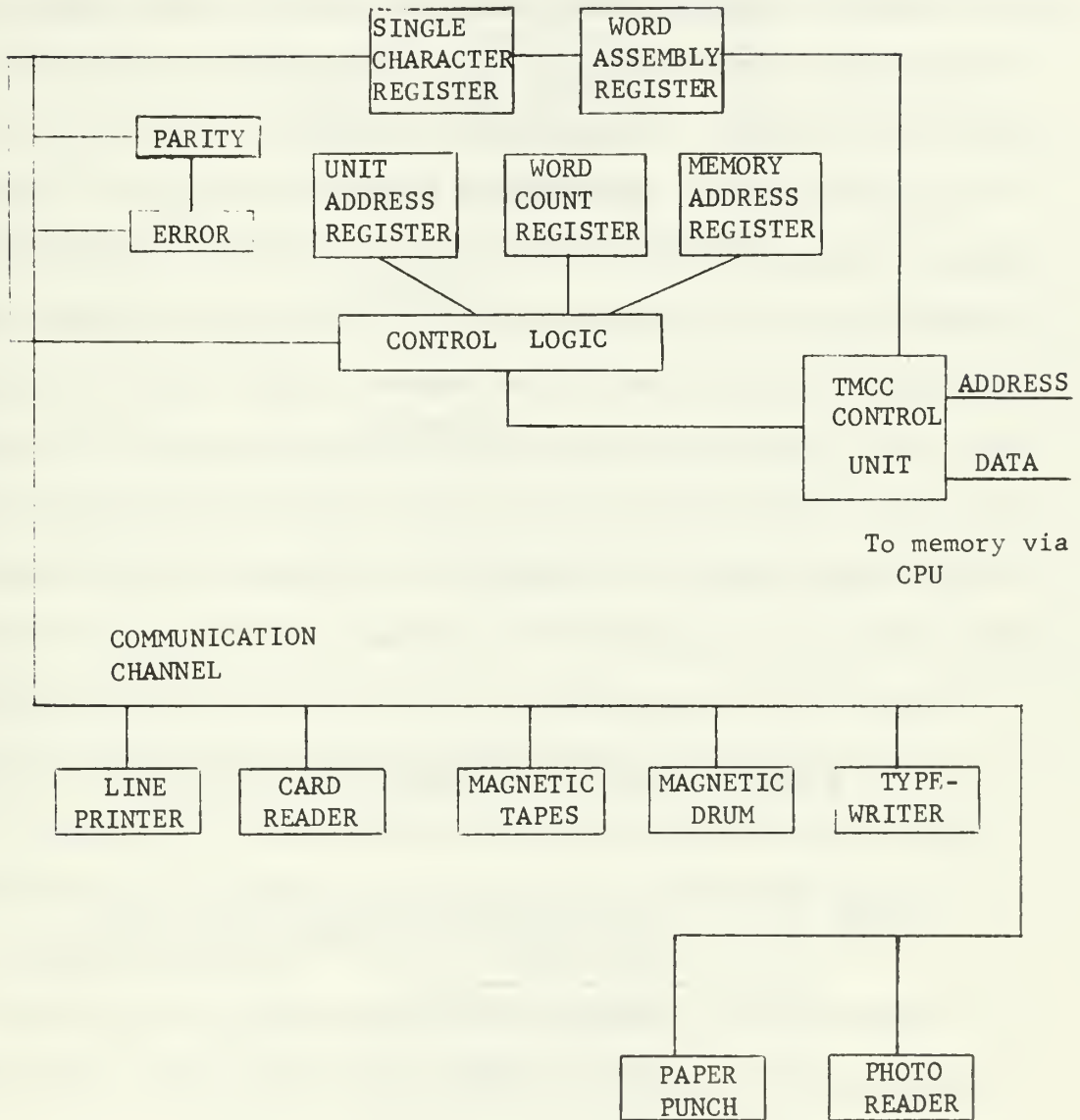


FIGURE 4



GRAPH OF RELATIVE CPU STALLING FOR ONE

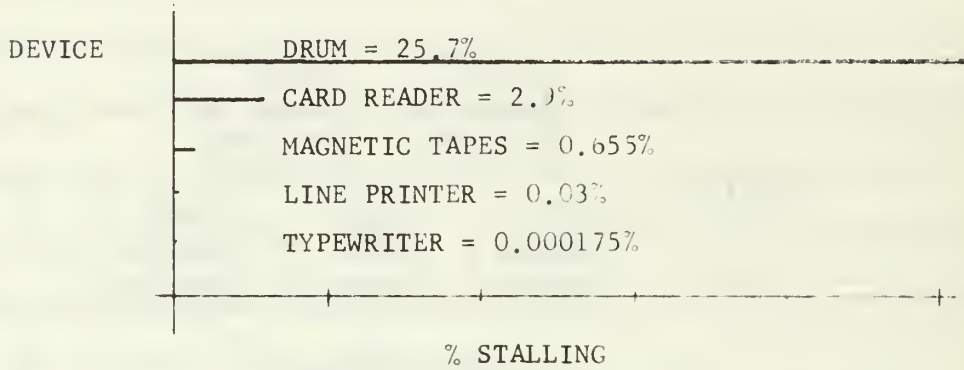


FIGURE 5

GRAPH OF RELATIVE CPU STALLING FOR TWO TMCC'S

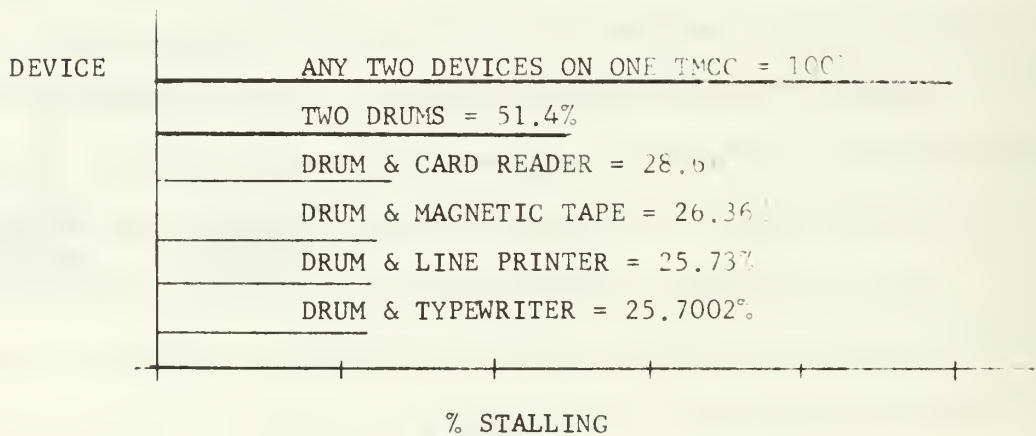


FIGURE 6

characters per line, line printer; two, 200 bits per inch, 15K character per second magnetic tape units with unit number selectivity from zero to seven; a two million word, 147K words per second drum; a 10 character per second, character teletypewriter in conjunction with the main control console and a 210 cards per minute card reader all on one TMCC (A).

The system capabilities are ideal for time sharing applications. The I/O channels are capable of data movement in parallel with CPU processing. The multiplexer allows memory access for more than one channel between CPU memory accesses. The bulk memory device is very fast, capable of swapping the entire core memory at a rate of 276 times per minute. However, because each channel may have I/O on one device at a time the advantages of parallelism are nearly lost. Swapping or paging activity must time share the single channel with active I/O resulting from execution of a program in the batch queue. This results in an unavoidable complete, or 100 percent, stalling of either the batch execution at I/O time, or the swapping activity. The installation of a second interlaced TMCC with only the drum on it, will significantly reduce the stalling from 100 percent to between the limits of 51.4 percent to 25.7 percent. (See Figure 6).

## B. SOFTWARE ENVIRONMENT

The computing system operates under control of the SDS Real Time Monitor (RTM). This system supervisor makes provision for controlling assembly and compilation, hybrid, display, and batch processing in an on-line, real time, simultaneous I/O mode. The RTM provides a set of debug routines along with a complete set of diagnostics for debugging and analysis of programming errors. All active programs are run in a batch processing mode referred to as background.

Background jobs are swapped out of core on to the drum if additional core memory is required for an interrupt service routine. After the interrupt is serviced, control is returned to the interrupted background program after it is swapped back into core. The areas on the drum where swapped programs are stored are reserved and protected by RTM from being written over.

Memory management is done by RTM. Figure 3 shows the subdivisions of core memory with the resident control portion of RTM in the lower section, interrupt processors in upper with some of the RTM's temporary pointers, and batch processes in the remainder of core. Non-resident RTM subroutines called by a batch process are added to the lower portion of memory moving the lower limit of the batch process higher into core. When a batch process is too large to fit in the core remaining after lower and upper are filled, RTM will not execute the batch program and gives an error diagnostic.

The system supervisor, RTM, has a resident processor, a Fortran IV processor, a Symbol and Meta-Symbol assembler, an overlay loader, an I/O processor and primary and secondary libraries.

The resident monitor processes all control messages and makes provision for users to write and define new control messages to specify additional activities or modes of operation. An interrupt processor (which saves hardware and program status) is included to control all interrupt processing. After an interrupt routine is serviced the interrupt processor restores the interrupted program and hardware status. The interrupt processor is re-entrant and interruptable allowing multiple interrupts to be processed without loss of the interrupt processor's arguments. The re-entrance program saves the arguments of

a subroutine if it is being re-entered as a result of an interrupt.

These arguments include local variables, temporary storages, return addresses, calling argument addresses, and are stored in a first-in last-out push down stack.

A resident loader is used to load all programs from the system files including the overlay file, secondary and primary libraries. The loader is semi-absolute and has the ability to search the system files for a called symbol at load time. When the symbol is located, the subroutine containing the referenced symbol is loaded and linkage is made to the symbol. Some resident supervisor subroutines are callable as subroutines for the general user which results in possible user program size efficiency.

The SDS Fortran IV processor includes an assembler, compiler and generates subroutines that allow real time operations. The Fortran IV language used is the standard machine independent mathematical language used in IBM and CDC systems.

The Meta-Symbol and Symbol assemblers translate machine language symbolic input into an octal semi-absolute format ready for compilation, loading and execution.

The overlay loader converts relocatable segmented programs into proper form for loading. Any programs that are segmented are loaded under control of the overlay loader which does the bookkeeping of when the proper time for loading is, and which segment is to be loaded next.

The system I/O processor is a subroutine that generates its I/O octal format from the calling arguments. This routine processes all supervisor and Fortran initiated I/O and with the proper calling



arguments it will process any I/O desired. This processor has built-in checks that control list movement and saves pointers to reserved files for later processing. All I/O is done on a first come first served basis with no I/O interrupt possible during an I/O operation. The interlace capability is used to do I/O in parallel with CPU activity.

The primary library is stored on the drum and consists of system routines, Fortran I/O, and routines to perform mathematical operations. Mathematical routines may be added as they are written and perfected by system users. The debug program is provided as a part of the primary library for tracing, patching, snapshots and program display.<sup>8</sup> The secondary library, which is also stored on the drum, includes interrupt services routines and batch production programs that are to be saved for production runs.

The system software is not designed for time sharing. Some of the subroutines required to correct this deficiency are available in the supervisor itself. The re-entrance subroutine is the most significant of the system software subroutines required for time sharing implementation. The resident monitor ability to process new control messages with arguments and the re-entrant interrupt processor are other important building blocks. The most significant improvement necessary is alteration of the assembler and compiler routines to include a re-entrant capability. The I/O processor's pointer tables and ability to reserve areas and hold the reservation are especially essential to a time sharing environment. The inclusion of a relocation processor for swapping and paging would complete the implementation of a time sharing capability.

<sup>8</sup> Scientific Data Systems, SDS Real-Time Monitor Reference Manual, P. 17-18, Publication 90 11 OGC July 1967



One of the most important features of time sharing software is swapping speed for paging through virtual memory, reading a page to be saved onto a drum, then writing a desired page into core. The nominal maximum word transfer rate for the drum is 147 kilohertz. When the system I/O processor is used to do the data transfer, the word transfer rate (see Figure 7) drops to 5.85 kilohertz, about 4 percent of the nominal rate. A simple high speed swapper (Appendix A) external to the system processor has an average word transfer rate of 85 kilohertz. (See Figure 8). The higher word rate is desirable in terms of response time for the system, however, the trade-off is loss of system protection of time sharing drum listings from batch and hybrid initiated writing activity, and vice-versa loss of protection of batch and hybrid drum listings from time share initiated writing activity.

The display console software is not adapted for source input of object programs. The existing software is especially adapted for input from the main control console and hybrid system for display of results. The lack of program independent linkage to the CPU makes its use for time sharing difficult, if not impossible with existing software. The display buffers are dynamic and relocatable and because of these features they are not consistently accessible for application of paging techniques. These factors complicate the requirement for linking debugging and editing capabilities from display lists to drum lists, and for orderly and rapid translation. Considerable study indicates that linking existing display software to the needs of a time sharing system console requires software patching more complicated, thus slower, than a new software package designed to do the time sharing task. The new software does not pre-empt existing

GRAPH OF TIME VS. WORDS TRANSFERRED  
UNDER RTM CONTROL

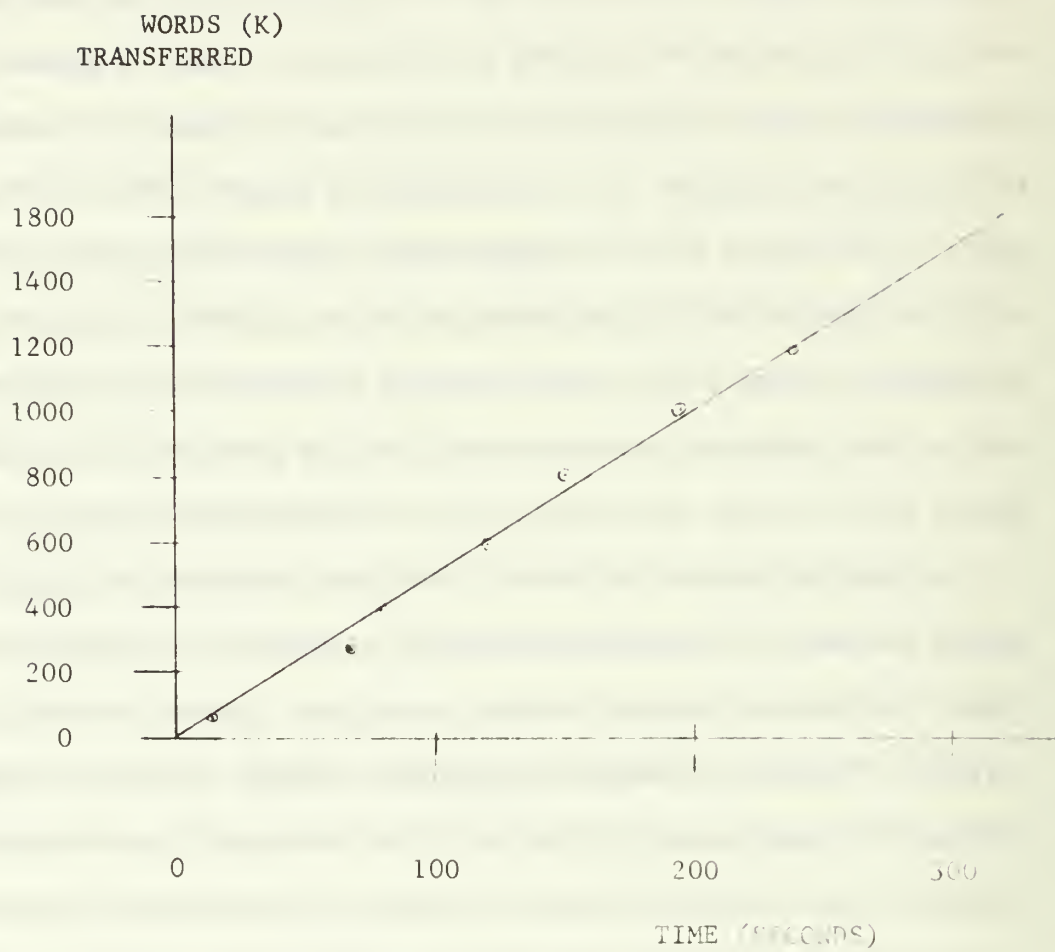


FIGURE 7

GRAPH OF TIME VS. WORDS TRANSFERRED  
UNDER HIGH SPEED SWAPPING DRIVER

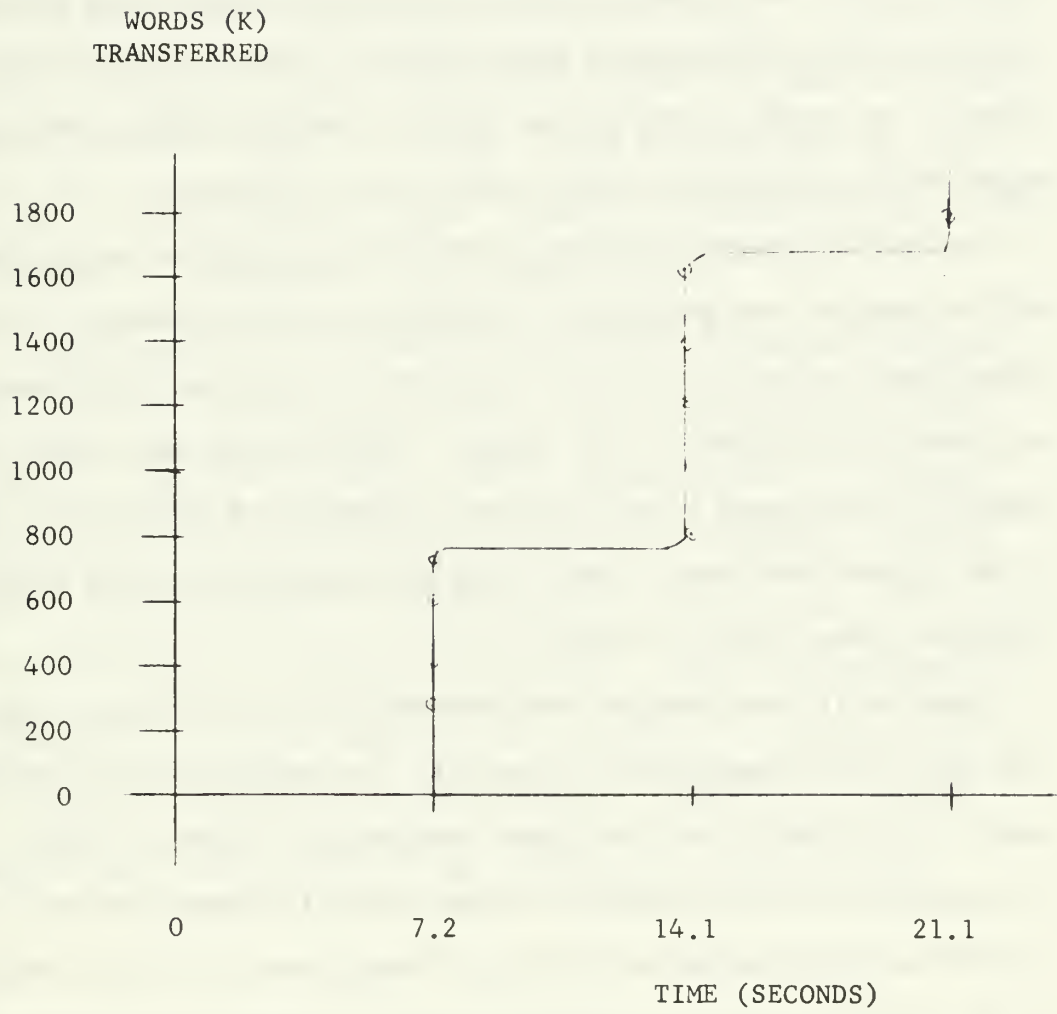


FIGURE 8

software capabilities nor uses, only makes possible a time sharing capability on call as an additional system subroutine.

### C. SYSTEM RELATION TO A TIME SHARING CONFIGURATION

The system outlined above has the basic hardware potential required for a time shared system. Most important is the multiplicity of user terminals. The main control console, each display unit, and the hybrid system provide stations where four potential users could share the system. The display consoles and the main control console are stations where on-line programmers would normally use the system.

The hybrid computing system and the display consoles access the digital computer for processing time with priority interrupts. These allow access to the CPU, hence, core memory at any time their particular priority is highest in the system. Provision has been made for additional interrupts to be installed to expand and restructure some of the system functions. These could be connected to a time sharing execution queue or put to other use.

Speed of I/O and swapping are provided by the interlace system and time multiplexed communication channels. The required fast swapping speed is provided by the high speed drum and the interlace TMCC. The interlace is very essential to the parallel movement of data on the TMCC during run-time on the CPU, if additional I/O is not required by the CPU. At least one additional TMCC is required to relieve the 100% I/O stalling problem.

The only hardware isolation provided is a set of manual protect switches built into the drum controller. This protection is needed regardless of whether the system is time shared or not. These switches could be set by an individual to save listings for execution

later if desired, however, the system supervisory software will save these listings until released.

The system software supervisor is not designed for time sharing. Because the assembler and compiler are not re-entrant these two functions are necessarily in the batch queue under a run until completion mode. When a program is assembled and compiled it may be swapped out to allow other users into the batch execution queue. Because there is not a relocation feature the program swapped out must be swapped into its original location in core. These indicate that for time sharing the batch execution area, a swapping procedure is best. This conclusion is supported by other system analysis.<sup>9</sup>

System provision for connecting additional interrupt routines, and swap-out of active batch routines during interrupt servicing, is used to save batch programs during the execution of interrupt programs that link to the time sharing swap-in, execute, swap-out cycles. The provision for retaining user listings after I/O is used to save listings for swapping on the drum. Additional software will be required to initiate swapping and do tracing of execution activity for interrupt linked time sharing programs. Routines to provide program input, and editing at the display are also required.



#### IV. PACKAGE PHILOSOPHY

The ultimate goal of the design of a special purpose time shared capability for the Electronics Engineering Computer Laboratory has several intermediate goals to be served in the achievement of a useable design. A desire exists to use as much existing software and hardware as possible. This reduces the problem of interfacing new programs and hardware to the old, reduces the additional core requirement for the larger supervisory program, increases speed by keeping the supervisor small and resident so supervisor swapping is not required, and will avoid costly bookkeeping errors that lose programs or user listings. The additional software should be stand-alone and callable as a sub-routine so that a "partially" time shared and foreground - background system will run under simultaneous control of the old and new supervisors. The time sharing capability should provide as fast as possible service to minimize mutual interference and time delays.

##### A. SYSTEM USERS

This computing system is used primarily by students who are learning the basic principles of hybrid computing techniques, or learning the basic principles of machine language programming and computer hardware capabilities. Hence, a considerable amount of time is spent doing editing, debugging, step-by-step execution, and rerunning of the same programs. Most of the basic hybrid programs require very little CPU time but monopolize the entire system by using the line printer as output in a print and calculate mode. The technique of step-by-step execution also monopolizes the entire system. In the case of step-by-step execution, the system is kept in idle except

during a step, when the system executes one instruction.

These two modes of operation are the most common used and the most wasteful of CPU time so any time sharing capability must allow at least these two types of users to share the CPU. The inclusion of a batch user into a group of step-by-step debuggers and hybrid users would be ideal. This special purpose time sharing supervisor is designed to provide service for two users doing step-by-step program debugging at the display console, one hybrid user, and one batch processor doing either Fortran IV or Meta Symbol production runs, with each user able to use the full capabilities of the system with a minimum of mutual interference. The users doing step-by-step debugging are allowed to rejoin the background queue for run time by any workable verbal agreement with the other users. The display consoles are not limited to step-by-step debugging. They are useable for on-line editing and correcting of successful or unsuccessful programs on a page-by-page basis and then re-entry into the run time queue by verbal agreement. Figure 9 is a pictorial outline of the envisioned special purpose time shared system with possible use configurations listed.

#### B. QUEUING

This computing system is installed in a unique environment that allows the execution queue to be entirely external from the hardware or software of the system. Because all four possible user stations are within conversational proximity of each other, the best and most foolproof method of determining execution priority is by word of mouth. This allows consideration of length of job; who is first in the queue; who the agreed primary user should be based on the urgency of the individual's time schedule and projected system use; which

# OUTLINE OF PROPOSED SYSTEM TERMINALS AND EMPLOYMENTS

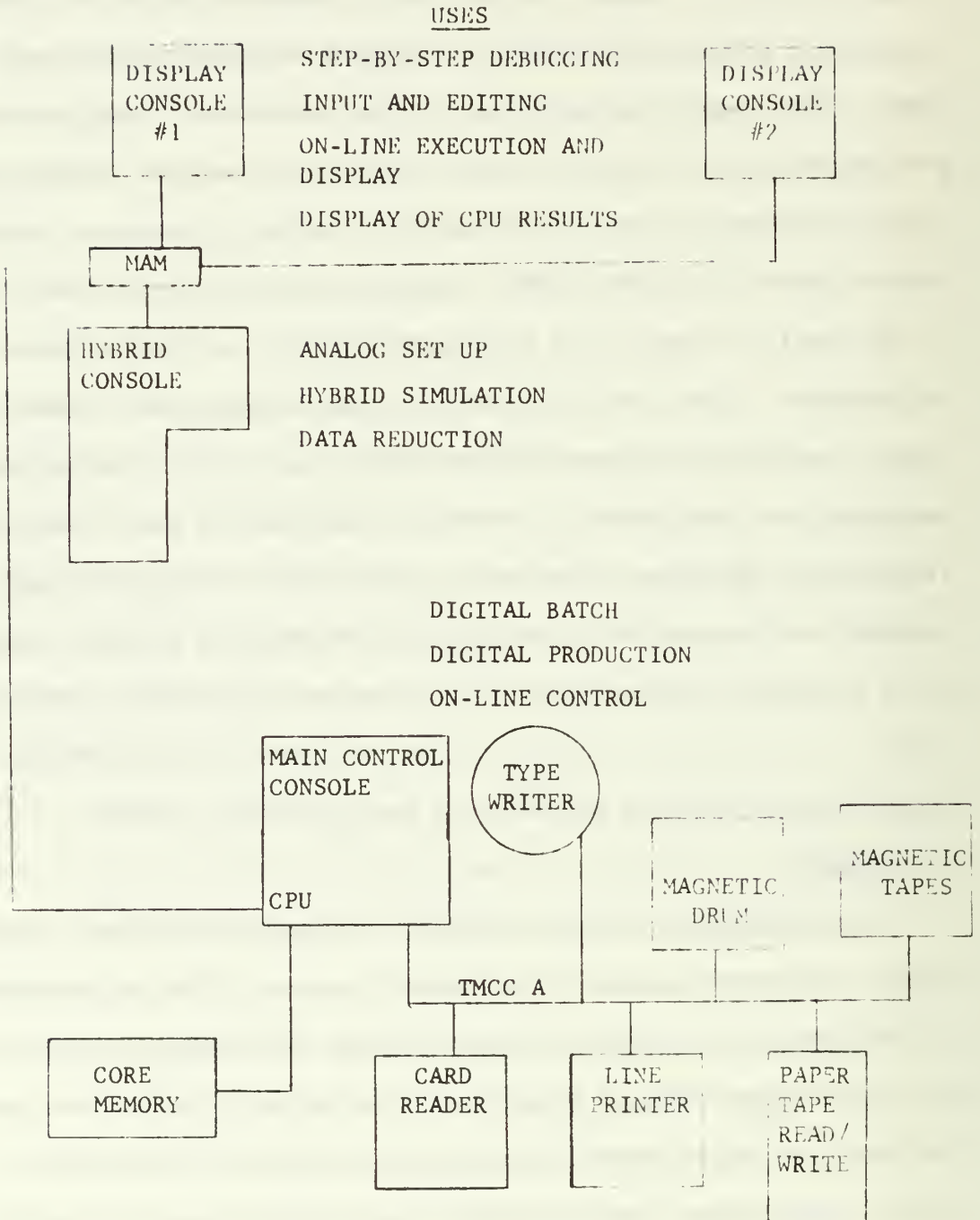


FIGURE 9

user is most likely to cause system disruption through destructive writing on the supervisory RTM; and of other less subjective basis for priority to be determined with the inherent adaptability of a man to man interface versus the inherent rigidity of a machine logic to man interface.

The run time queue is a verbally agreed on priority for batch or background\* execution of a user's program. This is distinct and different from the input, editing, debugging or step-by-step activity at the display consoles. These latter activities occur on an interrupt basis and are serviced in an internal queue (priority interrupts) on a higher priority than the run time queue. Limiting the run time queue to background operations allows the priority interrupts to be processed on a demand basis and avoids setting the system to an idle state for step-by-step debugging. In turn, the interrupt demands must necessarily be short and if at all possible unseen by the background operator to allow a reasonable program cycle rate for a group of background users.

The foreground\* user enters the run time queue by verbal agreement of his priority with the background users. This run time queue is an assemble, compile and execution sequence that ends for a program in this queue when execution is completed. The decision to operate in this mode, rather than enter background for execution from the displays in foreground via interrupts, was based primarily on the arguments discussed below.

\*Batch and background are used interchangeably.

\*Interrupt and foreground are used interchangeably and apply to users who are serviced by interrupts exclusively.



The system supervisor assembles, compiles and executes programs entered into the batch processor in background only. In the background mode RTM requires that at least one control message be input from the card reader or the control console teletypewriter during the job initialization phase. This requirement restricts foreground to background movement for any program to have at least one control message in the background area. Because background operations must be suspended for execution of a program input at the display consoles anyway, it was decided that the foreground user desiring to go to background execution should join the background queue and not disrupt the already formed queue with a priority interrupt entry. A second very important objection to interrupt entry into background is that the assembler, and compiler are not re-entrant. If a background job is interrupted for foreground entry into the background assembler and compiler during either assembly or compilation of an active background job, the interrupted job is lost and bumped out of the queue. Highly unsatisfactory for the bumped user!

The additional objections to priority interrupt entry into the background queue are less severe and could be eased with system development. The most obvious is the lack of multiple output media. Background users normally output on the line printer. Because only one line printer and TMCC are installed at present there could result a mixing of from two to four outputs if priority interrupt entry into the background queue did not provide a private output media. The most obvious private output media is the display console and software provision for this has been made. However, a hard copy, when desired, requires an additional entry and re-execution in the background queue at an inconvenience to the user.



Changing any portion of the RTM to provide re-entrancy to the assembler and compiler is not a trivial change. The RTM is a large interleaved software program with most of its links dependent on each other. There are several subroutines called by the assembler and compiler which would require changeover to a re-entrant capability. In addition, some of the subroutines in the assembler and compiler are called by other routines, and these individual internal routines would have to be changed to have re-entrancy too. The execution and check-out of such a change is an ambitious undertaking.

The hybrid user can operate entirely foreground and get nearly all the service he desires. However, the conflict over the single line printer as output device can arise. Most hybrid programs require some form of output listing of activity for evaluation of problem progress. The background processor also requires a copy of his results. The use of a verbal agreement to establish output priority is the best solution. In some cases a hybrid user must have output immediately so he can react to developments in a real-time simulation. In this case if large amounts of data are output at a single burst the line printer must be used. However, if only a small amount of data is required, then the teletypewriter is a suggested alternative. The teletypewriter is not a useful output device except for short, two line messages because its writing speed is much too slow (ten characters per second). An alternative is using the magnetic tape as output media for the batch processor, with a first-in first out queue on output from magnetic tape to the line printer when the hybrid system is in an activity null and the background queue is either empty or interruptable.

### C. HUMAN FACTORS APPLICATIONS

Applying the concepts expressed by some human factors engineers<sup>5, 6</sup> response times were designed into the software to be constant for each particular mode of operation. The step-by-step execution mode is one of the slowest at six seconds per step, however the user has several interpretive functions to perform at each step, hence a slower response time is tolerable. Paging is a slow process but is still easily tolerable at six seconds per page. The on-line programmer performing editing functions works on a page at a time and does not normally page rapidly through a program. The editing function response times (light per strike, character input, and line erasures) are less than the ten millisecond minimum response time of the on-line programmer, hence are adapted to continuous uninterrupted use at the on-line programmer's maximum ability. The only sizeable delay is after entering the background run time queue. This delay is either short in the order of 1 or 2 minutes, or long -- ten minutes or more, depending on the system activity. These times fall into the two categories of too short to be inconvenient or long enough to allow the on-line programmer to "time-share" his time between the run time queue and a secondary task.

The display presentations are formatted to the user's past experience as much as possible. This familiarity helps provide a better fit of the machine to the user. In the input, edit, and debug modes the scope face has an underlay that duplicates the standard Fortran IV coding sheet used for program writing. Because the programming activity is the same, the identical format helps the user feel familiar with his new environment and concentrate on the old set of rules he always has used for program writing. When the

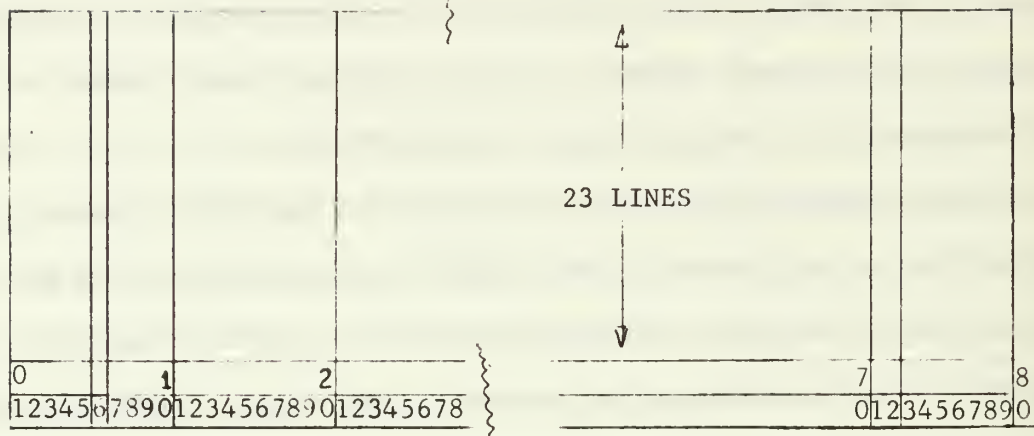
on-line user is in the step-by-step debug mode the scope format duplicates the main control console layout as nearly as possible. This format is modified to be read more easily than the control console by presenting all data as octal numbers in the same respective locations as seen at the control console, with the register labels linked to the contents with an equal sign. (See Figure 10).

A very important factor in the design of any software system is the feeling of confidence in the system a user gets when using it. The feeling of confidence is generated when his errors are either overlooked or interpreted as an erroneous action that can be undone if necessary. Because the human participant is the error-prone link in a man-machine system the machine portion must be designed to correct or ignore errors made by the human. The easiest method of achieving this versatility is to design the software so that the machine will present the fact that an error has been made and name the error for the human user to evaluate what his correction action will be. Likewise, it is very important that the user cannot cause the software to write on itself or lose its place by making a mistake at his console. Most operations must be done in a particular sequence, otherwise they could confuse or destroy the supervisor. An attempted execution out of sequence should initiate a warning and abortion of erroneous action.

#### D. ENGINEERING TRADE-OFFS

The trade-offs made in the design of this system were dictated principally by the existing system and the intermediate goals of the system design. Such things as sacrificing speed for isolation in the swapping routines by using RTM control, verbal entry into the

# DISPLAY SCOPE FORMAT FOR INPUT AND EDITING



# DISPLAY SCOPE FOR STEP-BY-STEP DEBUGGING

HALT		
F=	P=	I =
A =		B =
		X1 =
		X2 =
		X3 =
ERROR MESSAGE:		
ARG1 =		ARG2 =
ARG3 =		ARG4 =
ARG5 =		ARG6 =

FIGURE 10



background run time queue versus interrupting background execution for a priority entry, assignment of output media by verbal agreement instead of additional software controlled queuing and rigid assignment, and requiring additional good programming techniques of machine language programmers are the major trade-offs. These do not significantly slow the speed of servicing and they maintain the true strength of the non-time shared system, that is, adaptability. The system configuration can be changed with control messages by each user to fit his particular requirements and not violate other system users sharing the CPU. The only limits are essentially timing and interference and these can be worked out on a verbal level more logically than by a machine.

#### E. RESULTANT DESIGN OUTLINE

The proposed flow of program execution is important in the design of the special purpose time sharing system. All programs input must be processed in the background area for assembly and compilation prior to execution. Interrupt users initially run their programs in for assembly and compilation in the background run-time queue, then save them on the secondary library. Once input, interrupt programs are executed on priority interrupt demand. Whenever an interrupt occurs the background activity is suspended until the interrupt is serviced, then background activity resumes. The interrupt could result from a hybrid call for digital computation or I/O, a display character input, a display light pen strike, a display end of display buffer, a swap-in and execution of one step and swap-out at the display, a special purpose function like a line erasure on a display, or an iteration of an on-line program after a data change at the display. Programs are typed in and edited at the display



consoles entirely on interrupt servicing. To execute these programs the input is switched on to magnetic tape and from magnetic tape executed in the batch run-time queue. Programs input to the display buffers for debugging and editing are executed by switching the edited program to magnetic tape and executing in the batch run-time queue.

The software functions that meet the above specified time sharing system design requirements and fit into the existing system environment are outlined below.

1. Program input at Display -- Processes display interrupts to provide program input and transfer from foreground to background for execution in the background run time queue.
2. Program input from the Card Reader (C/R) to the display editing processor -- Transfers of a program read in at the C/R into the foreground for paging and editing to correct syntax and logic errors.
3. Paging Processor -- Provides a means to page through bulk memory and restore the edited pages in core back into the bulk memory.
4. Display execution routine for octal programs -- Swaps in, executes and swaps out octal programs, then provides for program data changes, and a re-execution or termination option.

## V. SOFTWARE DESCRIPTION

The special purpose time sharing package will provide two additional basic types of service during hybrid and batch processing:

1. Program input, editing and execution in foreground and background respectively, on an iterative basis.
2. Step-by-step execution with a tracing of program activity.

The subroutines required for these two additional functions on a time shared basis during hybrid and batch processing are outlined below. Discussion of program linkage and of the programming principles adhered to, with the trade-offs involved is included. A detailed discussion of the subroutine features with flow charts of program logic follow.

### A. PROGRAMMING PRINCIPLES

Correlation of system goals to good programming principles yields a set of rules to adhere to when writing software. In this time sharing application speed is of prime importance. A close second is minimal size of software. The original system design set limits of less than 8K words available for a time sharing supervisor. All of the subroutines are of a re-entrant nature so that one subroutine will service all requests of a common functional nature. This was accomplished by using indexing to point the proper argument in an argument table. Each subroutine saved its return address and had an argument table. All interrupt service routines saved the registers and portions of core they used and restored them as part of a standard end-action routine. To increase the speed of code translations the method of table look ups with the table base address indexed by the

entering argument to get the location of the exit argument was used. Whenever possible, registers were used for temporary storage because of the high speed of register data transfer versus slower memory storage. Maximum use was made of the single cycle register masking and manipulation instructions. All I/O was done under interlace control, with the exception of the drum I/O which was done under RTM control to insure saving of drum lists. Use of system subroutines was limited to only situations where linkage had to be provided to save common arguments. Because of the interdependence of the RTM subroutines many extra checks and tests are built-in but not required by simpler routines such as many used in this software package. These extra checks absorb sizeable blocks of time.

## B. FUNCTIONAL DESCRIPTION

The following descriptions of the subroutines used for the two basic types of additional service are of the function provided, with no specifications of how. The basic interrupt routines service both the program input, edit, and debug mode and the stepping mode and are included in this discussion.

### 1. Program Input Editing and Debugging

The information flow and functional organization of the subroutines to perform the input editing and debugging function is outlined in Figure 11. The subroutines required to perform this service are listed below:

- a. `\\KEYD` - Entered by use of a control message with a single field specification that indicates which display scope is desired. The re-entrant index pointer is set to the proper arguments. The desired display is started and its lists

FUNCTIONAL ARRANGEMENT OF INPUT,  
EDITING AND DEBUG SUBROUTINES

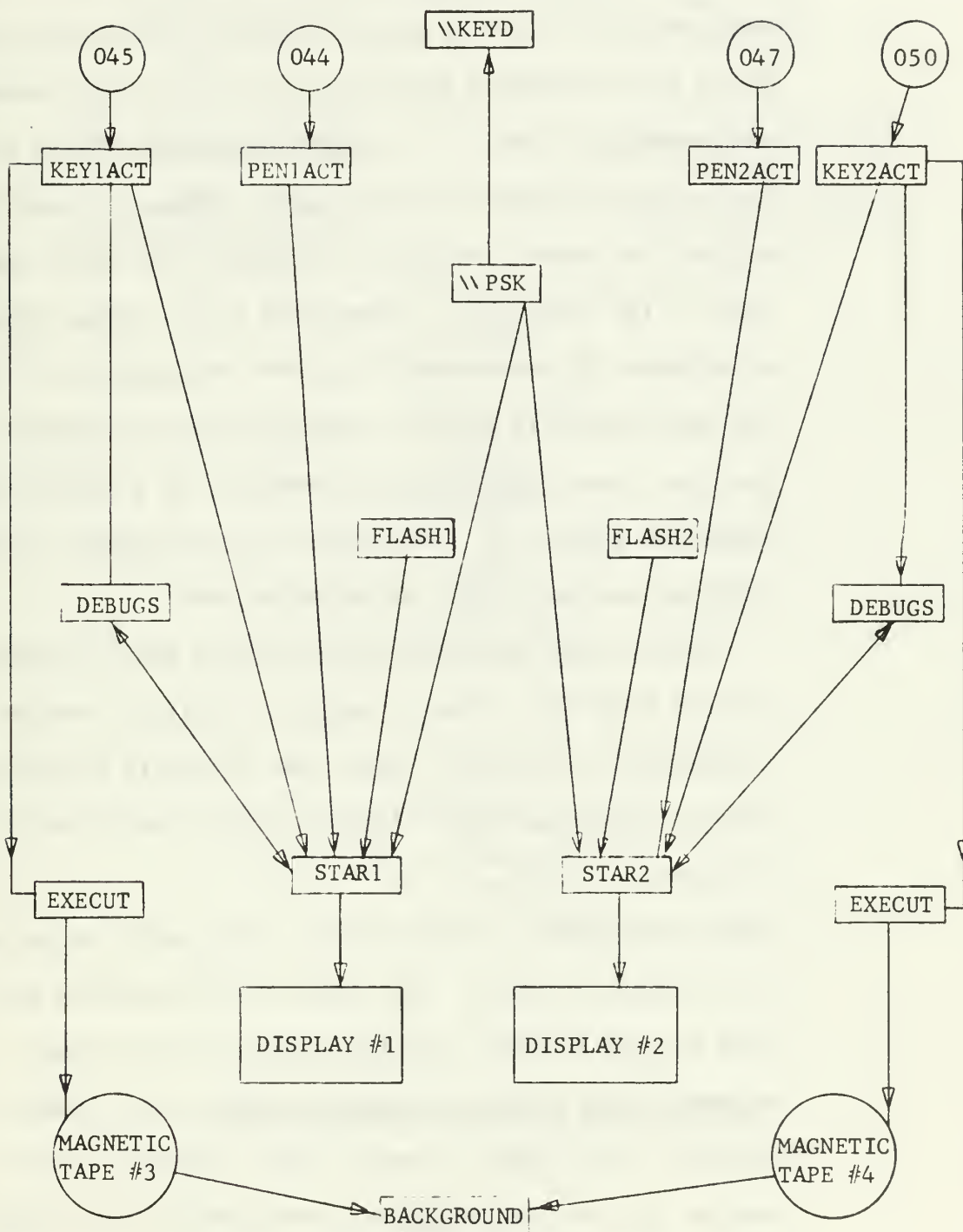


FIGURE 11

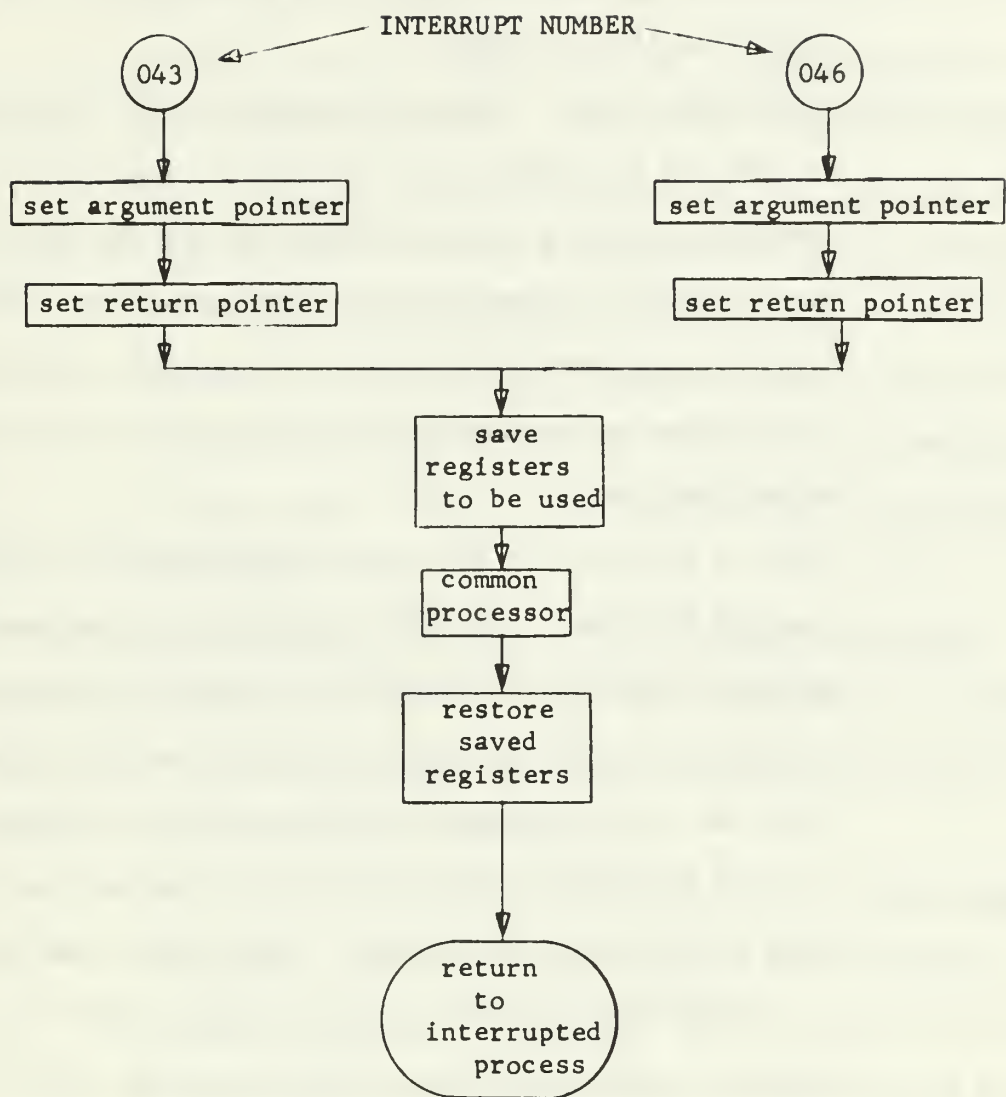
are cleared. All input pointers are zeroed and the display is set up for input.

- b. KEY1ACT AND KEY2ACT - These are the entry point subroutines for keyboard or function panel interrupts. The entry routines record the interrupted address and at which display console the interrupt occurred. The index register is set to point to the proper argument in the argument tables for re-entrancy and then the common processor is entered. The octal code input at the interface is interpreted in the common processor to determine if the interrupt was from the keyboard or function panel and what action is required of the software; character input, special typing function, or a function panel subroutine execution. See Figure 12 for an example of the interrupt service program organization used.

Several major actions are initiated by special typing function panel key strokes. Paging for editing, erasure of a designated or partially input line, and entry from foreground to background are initiated by this subroutine in this mode.

- c. PEN1ACT AND PEN2ACT - These are the entry point subroutines for a light pen strike. They record the interrupted address and set an index register to point to the proper argument in the re-entrant argument table of the common processor. The common processor inputs the word in the display list and the character corresponding to the strike is set to zero so the strike location is displayed as a delta ( $\Delta$ ). Other pen strikes are inhibited until a





INTERRUPT SERVICE ROUTINE GENERAL FORM

FIGURE 12

correction character is typed in at the keyboard. A second action is allowed, an erasure of the entire line the strike occurred in. The light pen is not enabled until the entire correction line is input.

- d. FLASH1 AND FLASH2 - These are the entry point subroutines for the end of display list interrupts. They save the interrupted program's return address and set the pointer to the re-entrant argument table of the subroutine. They then branch to the common processor which restarts the display that stopped and returns execution control to the interrupted program.
- e. DEBUGS - Entered from the keyboard processor by stroking the "ALT MOD" key. The mode of operation is switched to editing by the ALT MOD key with the light pen, character keyboard, and function panel enabled for editing. Because the light pen needs a signal to occur, a check (✓) is presented instead of blanks in the text on the scope so blanks may be altered. Exit to the input mode is accomplished by stroking the functional panel key labeled 1. After exiting the last page to be input is displayed and the pointers point to the next character position.
- f. \PSK - Entered by a control message with a field specification that names the display console to be used. This routine initializes the display, then reads a source program in at the card reader and stores it in the named display drum buffers. The program is then assembled, compiled and executed in the background queue.

- g. EXECUTE - Entered by typing " GO" and a carriage return on the display scope keyboard. This subroutine reads the source program in the display buffers and drum page buffer onto the magnetic tape. The program can now be read as source input in the background queue. This effectively inputs a source program into the background queue from the foreground area.
- h. STAR1 AND STAR2 - These are fixed position, fixed length display buffers. Each display has a private buffer area that allows for 23 lines of small characters to be viewed. The buffers also include the Fortran IV coding sheet underlay for input mode. The lists are self-chained to avoid flicker problems.

## 2. Step-by-Step Execution with a Tracing of Program Activity

The flow of information and basic functional organization of this activity is outlined by Figure 13. The subroutines required to perform this service are listed below.

- a. STEPBY - This subroutine is entered with a variable number of arguments; the display number, and a list of program variables desired to be traced as the first step in the calling program. The maximum number of variables to be traced is six. Specification of more produces six, or less, in an unpredictable fashion. The calling program is read onto the drum for future reference and the display is cleared and initialized to the control console format. All the initial register and variable values are saved for the stepping phase of the program. Control is returned to the calling program for execution.

# FUNCTIONAL ARRANGEMENT OF STEP-BY-STEP EXECUTION SUBROUTINES

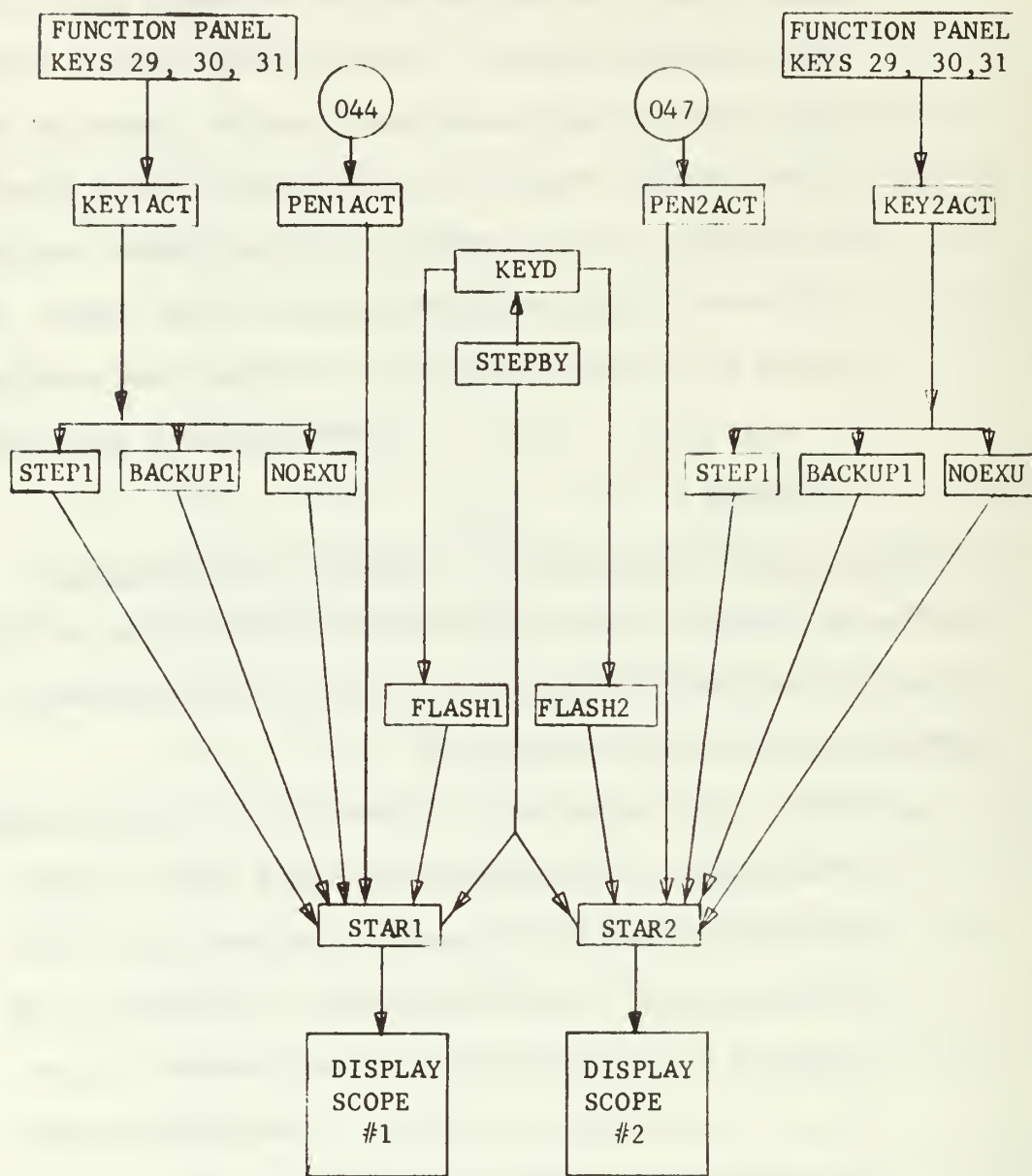


FIGURE 13

- b. STEP1 AND STEP2 - Before entry to the common processor a flag is set in STEP1 or STEP2 to indicate which display is using the program. The argument pointers are already set by KEY1ACT or KEY2ACT. Action is initiated by stroking the step button (Key 31) on the functional panel. The common stepping routine interprets one displayed instruction and executes it, then updates the displayed registers and variables and returns to the end action routine. One instruction is executed after each stroke of the step button. Provision is made to interpret changes to the displayed registers (with the exception of the F and P registers) and execute the changes. The purpose is to simulate the main control console capability during step by step execution.
- c. BACKUP1 - Entered from KEY1ACT or KEY2ACT resulting from a stroke of Key 29 on the functional panel. This decrements the P register once and displays the previous instruction. No other displayed registers or variables are affected.
- d. NOEXU - Entered from KEY1ACT or KEY2ACT as a result of a stroke of Key 30 on the functional panel. This increments the displayed P register once and brings in the next instruction without executing the instruction in the display instruction register.
- e. KEY1ACT AND KEY2ACT - The step-by-step routine is controlled from the function panel, therefore the required interrupt saving and re-entrant argument table pointer setting are done by these calling subroutines. Alterations



of the displayed registers are input at KEY1ACT or KEY2ACT but processed and stored under PEN1ACT or PEN2ACT. The restoring of interrupted registers, return to the interrupted background process, and other appropriate end action activity is also executed in KEY1ACT or KEY2ACT.

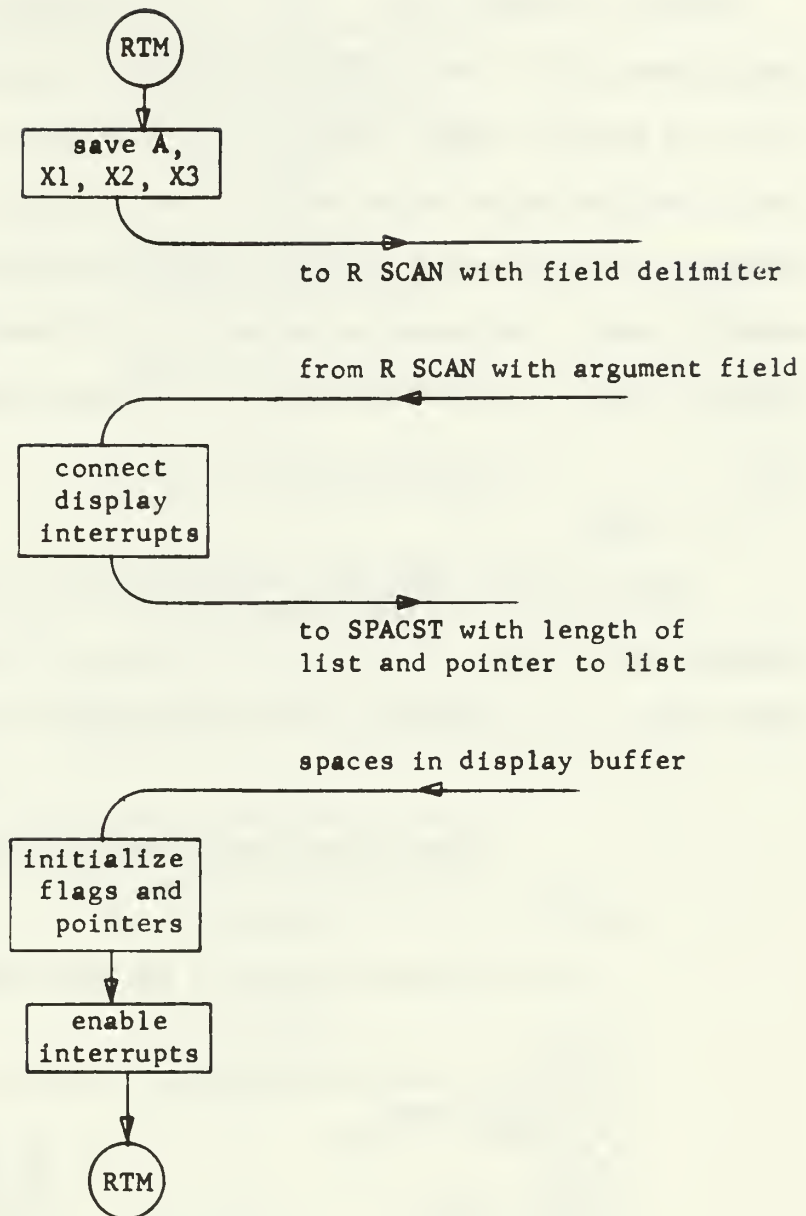
- f. PEN1ACT AND PEN2ACT - Same as above with the exception that the erasure of the entire line the light pen strike occurred in is inhibited.
- g. FLASH1 AND FLASH2 - Same as before.
- h. STAR1 AND STAR2 - Same as before, but set to display the main control console registers.

#### C. SUBROUTINE DISCUSSION AND LOGIC

The subroutines of the special purpose time sharing package are here broken down into their discrete functional components and have their logic traced with corresponding flowcharts.

##### 1. \KEYD

"\KEYD" is entered from RTM, initiated by a control message " $\Delta$ KEYD (arg.)" where "arg." is the entering argument specifying which display scope is to be initialized, 1 or 2. The registers to be used are saved to preserve the system supervisor arguments. The control message argument is entered using RTM. The display number is used in an index register to point to arguments used by this routine that apply to the named scope. The interrupts of the named display are connected to the time sharing interrupt processor and the display is started. The display buffers are cleared, then the pointer and checking flags are set to start input of new lists. The Fortran IV coding sheet underlay is initialized for display. After the registers used by \KEYD are restored, the display



SUBROUTINE \\KEYD

interrupts are enabled for program input and editing activity and return to RTM is executed.

## 2. EXECUTE

Execute is entered from PSK to bring programs read into the foreground drum buffer from the card reader back into background by writing it on the magnetic tape. Execute is also entered by typing " GO" and a carriage return at the display console to put the program on the foreground display buffer into the background queue by writing it on magnetic tape. The execution sequence is outlined in Figure 14 for programs entered into the background queue under control of the display.

## 3. \\PSK

PSK is entered from RTM. The entry is initiated by a control message " $\Delta$  PSK" (arg.)" with arg. the number of the display whose buffer area is to be filled. The display initiation routine (\\KEYD)

Typed at the Display Console Keyboard:

$\Delta$  GO

In the Background Queue in the Card Reader:

$\Delta$  JOB

$\Delta$  ASSIGN SI=MT3A for Display #1, MT4A for Display #2

$\Delta$  ASSIGN (args)

$\Delta$  META9300 (args)

$\Delta$  EOF

$\Delta$  LOAD (args)

$\Delta$  EOF

}  
next background job

or  $\Delta$  FORTRAN (args)

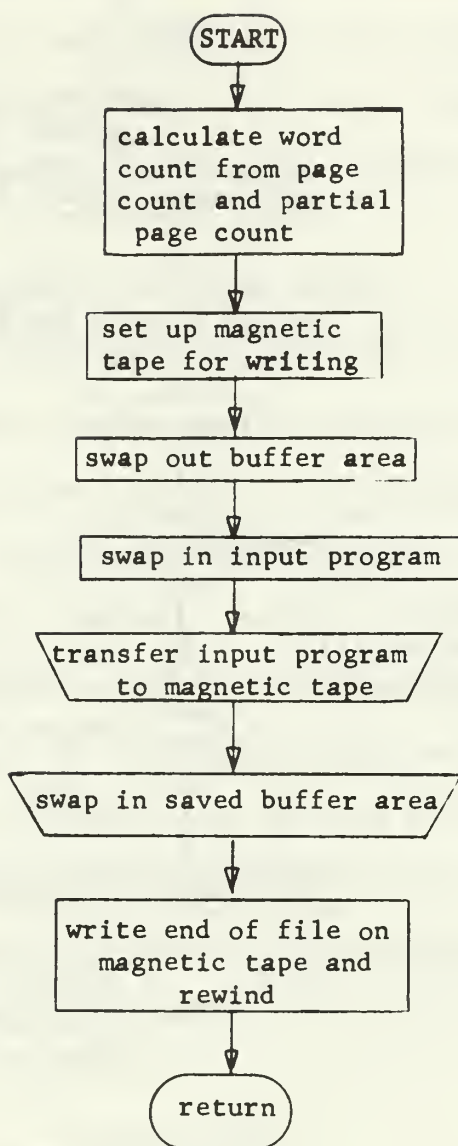
$\Delta$  LOAD (args)

$\Delta$  DATA

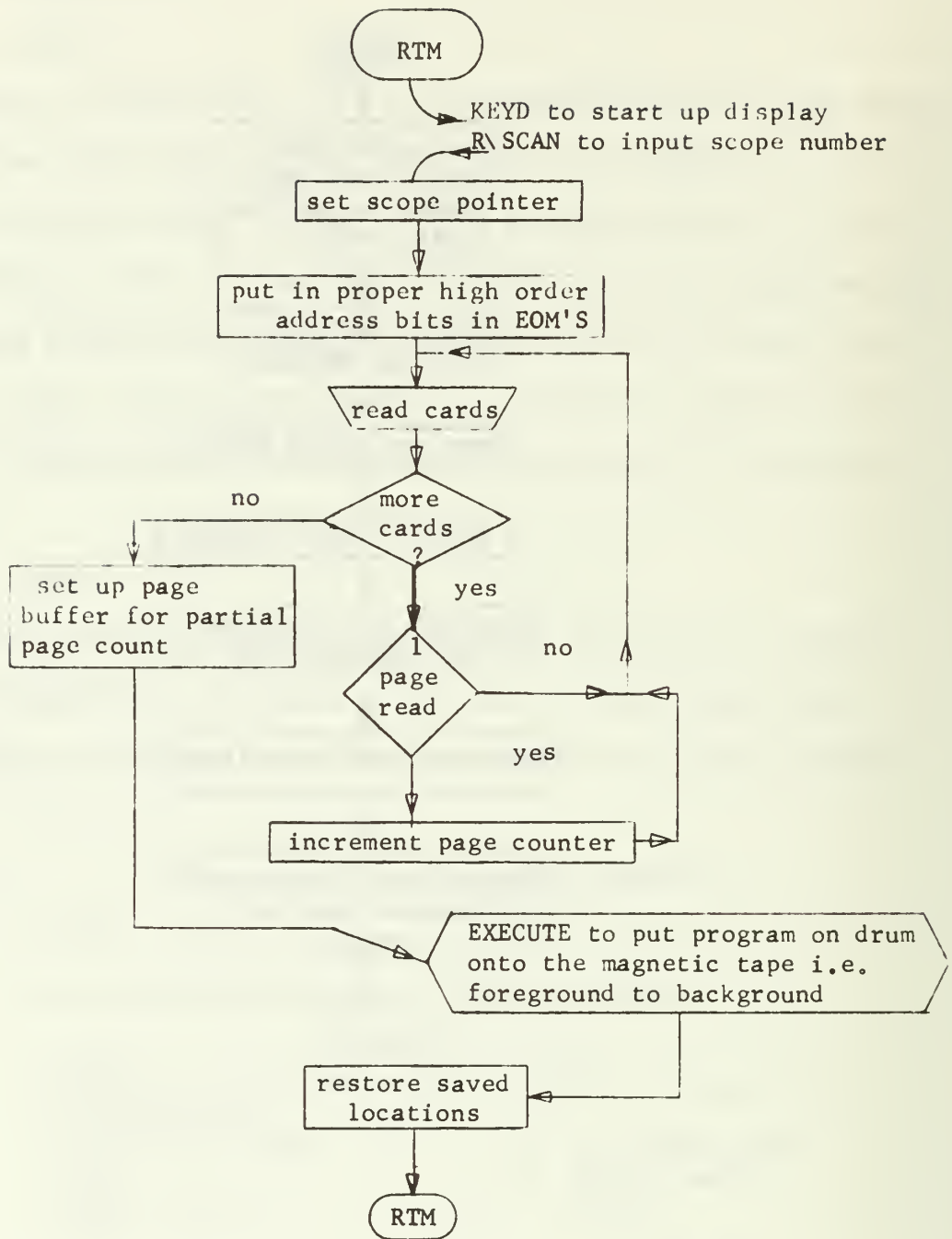
}  
data deck

}  
next background job

Figure 14



SUBROUTINE EXECUTE



SUBROUTINE \ \ PSK



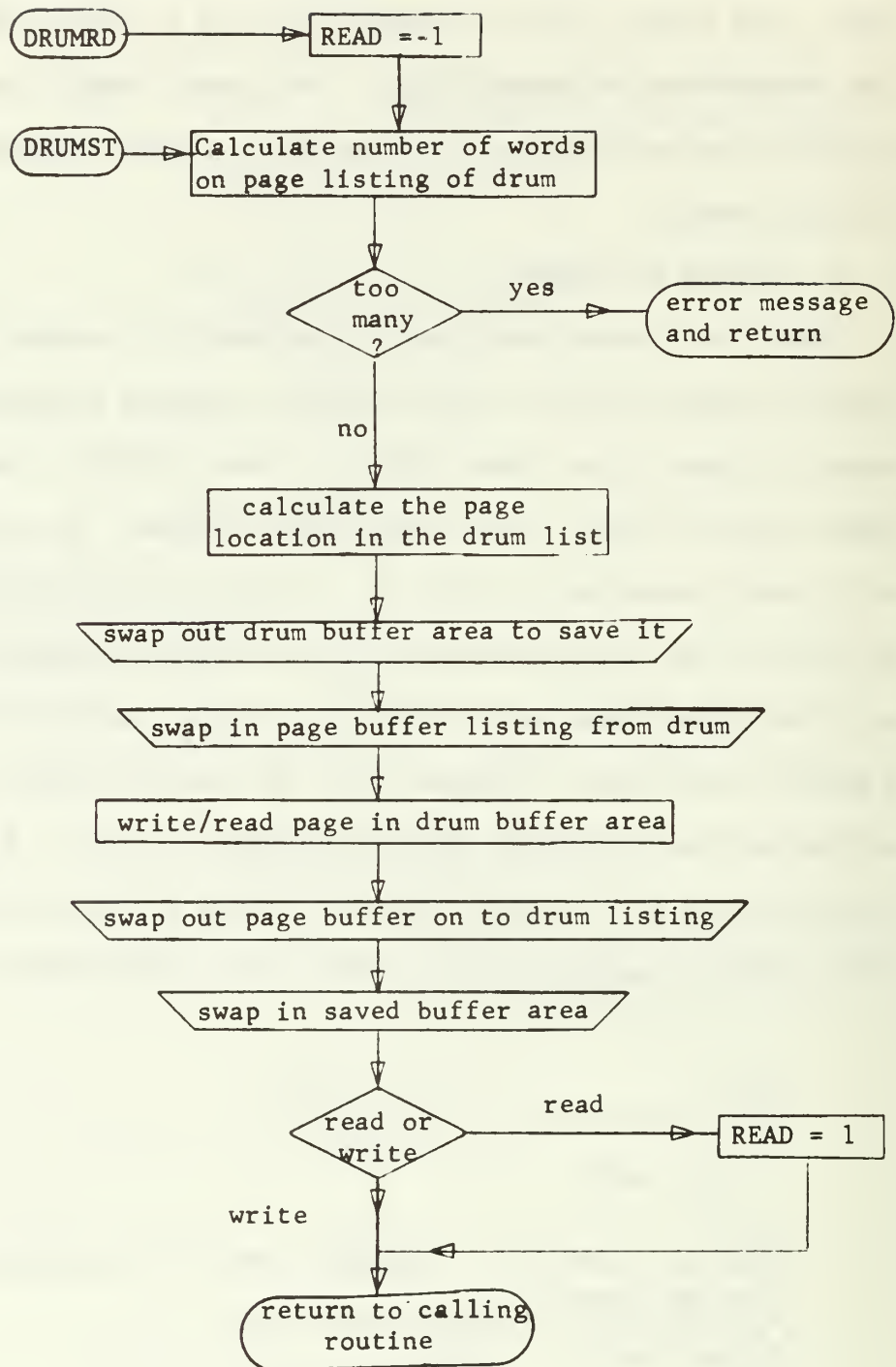
is called for setting up the display. The paging counter (FACES) is reset and the high address bits set in the energize output (EOM) instructions. The cards in the card reader are read in and saved a page at a time on the drum and magnetic tape. The source input is read from magnetic tape for execution in batch mode. Figure 15 shows the proper calling sequence.

#### 4. DRUMRD and DRUMST

These are entered from other subroutines that request paging of blocks of memory in and out of core. The entering argument is the number of pages to be stored (FACES) or read (ENDING). For reading pages from the drum a flag is set (READ) negative. The common processor is made re-entrant by the prior setting of an index register (X1) to point to the correct arguments in the argument tables. The pointer to the proper page on the drum is calculated and stored (BUILD). A buffer area in core is swapped onto the drum for saving, then the desired program listing on the drum is read into core. A page is transferred to or from the respective display buffer, then the program listing is swapped onto the drum. The saved buffer area is then

△JOB	
△PSK (1 or 2)	
object deck	
△EOF	
△ASSIGN SI=MT3A for Display #1, MT4A for Display #2	
△Assign (args)	
△FORTRAN (args) or △META9300 (args)	
△LOAD (args)	△EOF
△DATA	△LOAD (args)
}	△EOF
data deck	}
}	}
next job	next job

Figure 15



SUBROUTINE DRUMRD and DRUMST

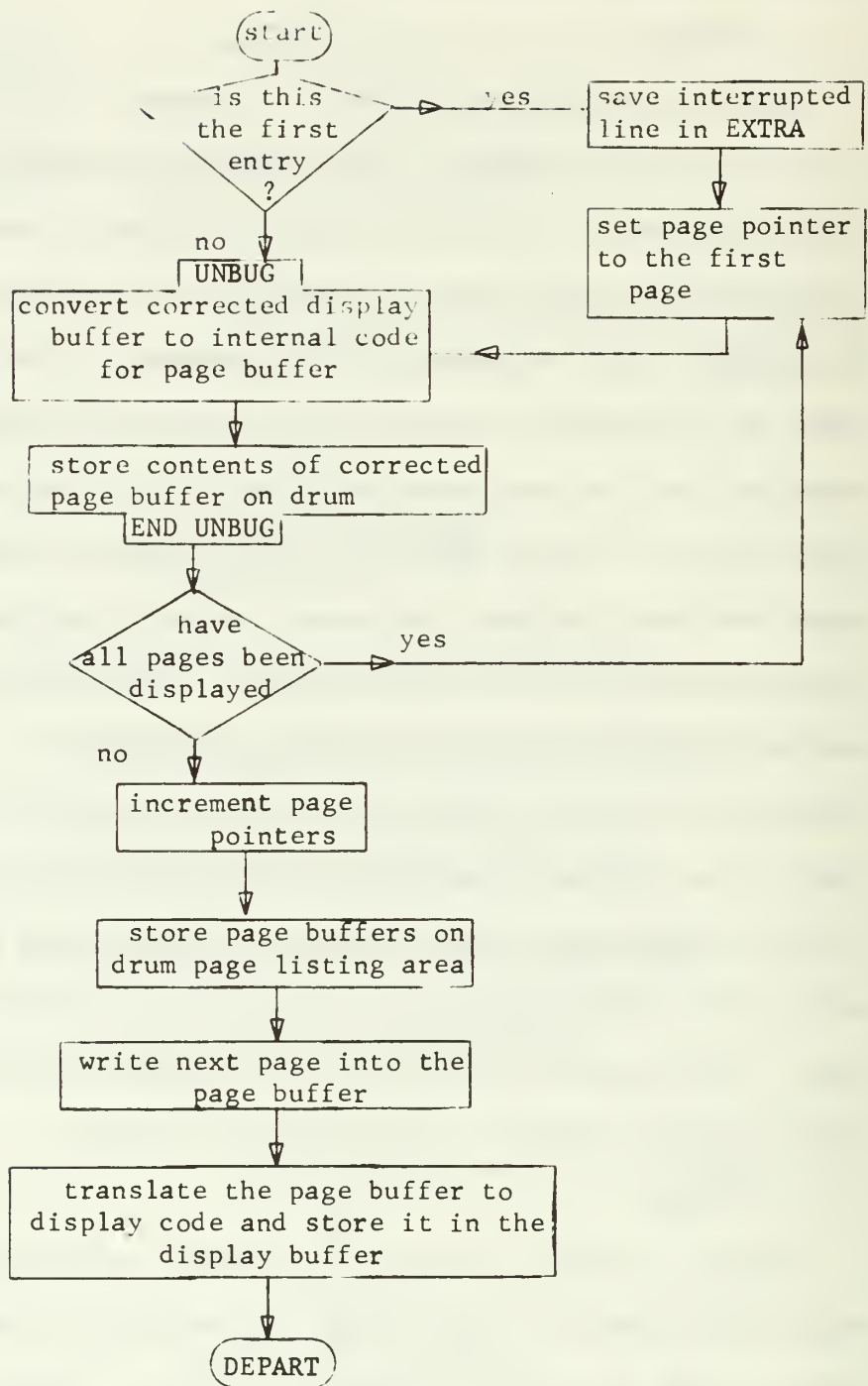
restored to core from a saving buffer on the drum and execution is returned to the calling program.

## 5. DEBUGS

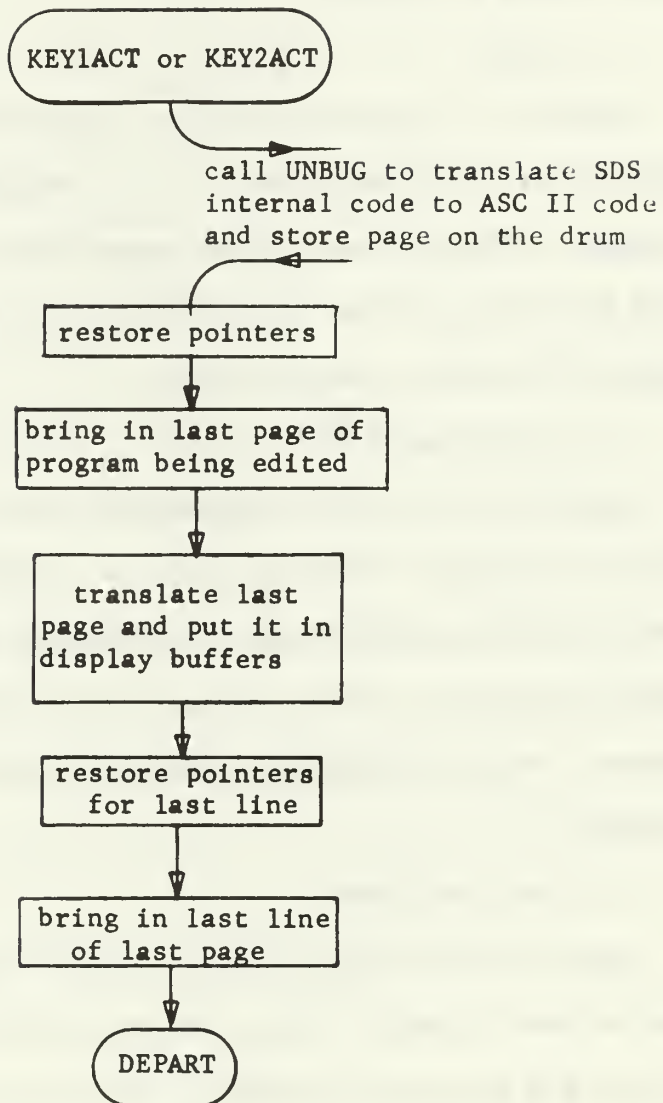
DEBUGS is entered from KEY1ACT as a result of striking the "ALT MOD" key on the display keyboard. This subroutine controls the displays in the editing mode. It saves the present contents of the display buffers and starts paging from the beginning of the program in the display buffer. Each succeeding page is presented on the scope. The light pen and keyboard interrupts are enabled for erasing erroneous characters or lines and inputting corrections to the errors. Successive strokes of the ALT MOD key displays successive pages. When all pages have been displayed, the paging pointers reset and start paging from the beginning of the program again. Exit from this routine is executed by striking the function panel key labeled 1. There are two buffers used, the page buffer, which holds the SDS internal code form of the program for swapping, and the display buffer, which holds the ASC II code form of the program for display. The program is held on the drum in SDS internal code ready to be transferred to background mode. During paging, the translations between the page buffer and the display buffer utilize a rapid translation program.

## 6. DDEBUG

DDEBUG is entered from KEY1ACT or KEY2ACT only when the keyboard and function panel are under control of the DEBUG subroutine. The entry is made by striking function panel key 1. DDEBUG returns the display to the input mode, from the editing mode. The display input pointers are reset for input of the next character on the line where the shift to editing occurred. The page on display in the edit mode is translated into the page buffer and stored on the drum. The page



SUBROUTINE DEBUGS



SUBROUTINE DDEBUG



that was stored when editing was initiated is translated and displayed. If a line was in the process of being input, and consequently was not on the page buffer, it is returned to the display scope. Then the keyact exit routine is executed.

#### 7. SPACST

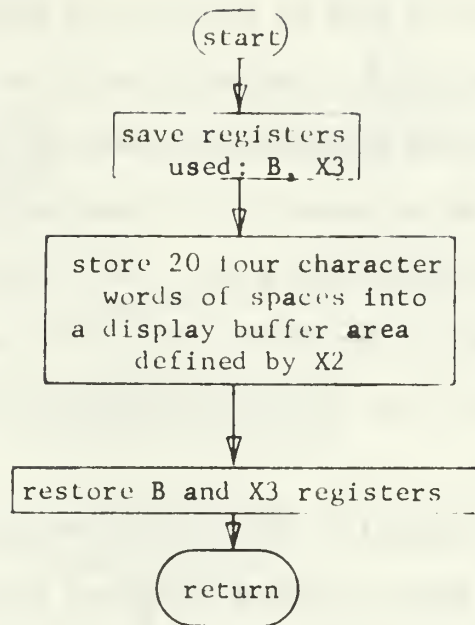
SPACST is a general subroutine used to store spaces in a display buffer area specified externally by an entering index register 2 (X2) argument. SPACST stores spaces twenty, four-character words at a time, into the display buffer specified by index register 1 (X1), and the line of the buffer specified by X2.

#### 8. FLASH1 and FLASH2

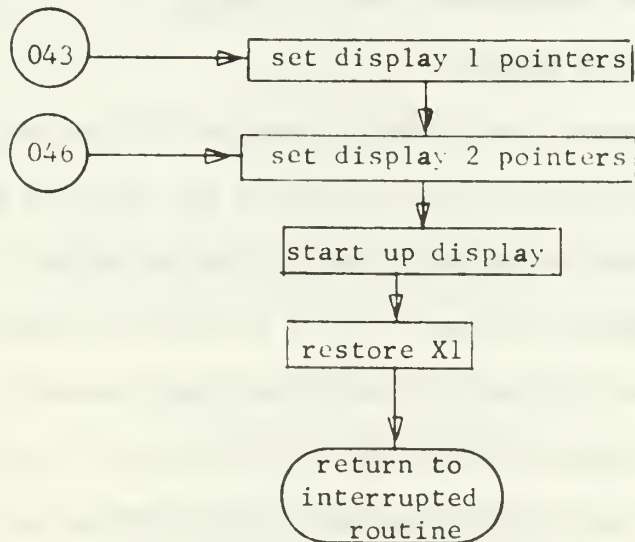
Interrupt 043 and 046 respectively, enter these subroutines which set index register 1 (X1) to point to the correct display argument in the re-entrant argument table. The display which initiated the interrupt by sensing an end of display list is started by the common processor. Then X1 is restored and return to the interrupted routine is executed.

#### 9. STAR1 and STAR2

These are the buffer and table subroutines. The display buffers, the Fortran IV overlay lists, the page buffers, the translation tables, and the EOM table are held here. The display buffers have address entry points for the error messages and the main control console register representation used in the step-by-step execution mode. The page buffers for the SDS internal code representation of the ASCII coded programs in the display are in this subroutine. The tables for translation between these two codes and the table of EOM instructions that control the display interrupt arming and transfers of data at the interface are held in this subroutine also.



#### SUBROUTINE SPACST



#### SUBROUTINE FLASH1 - FLASH2

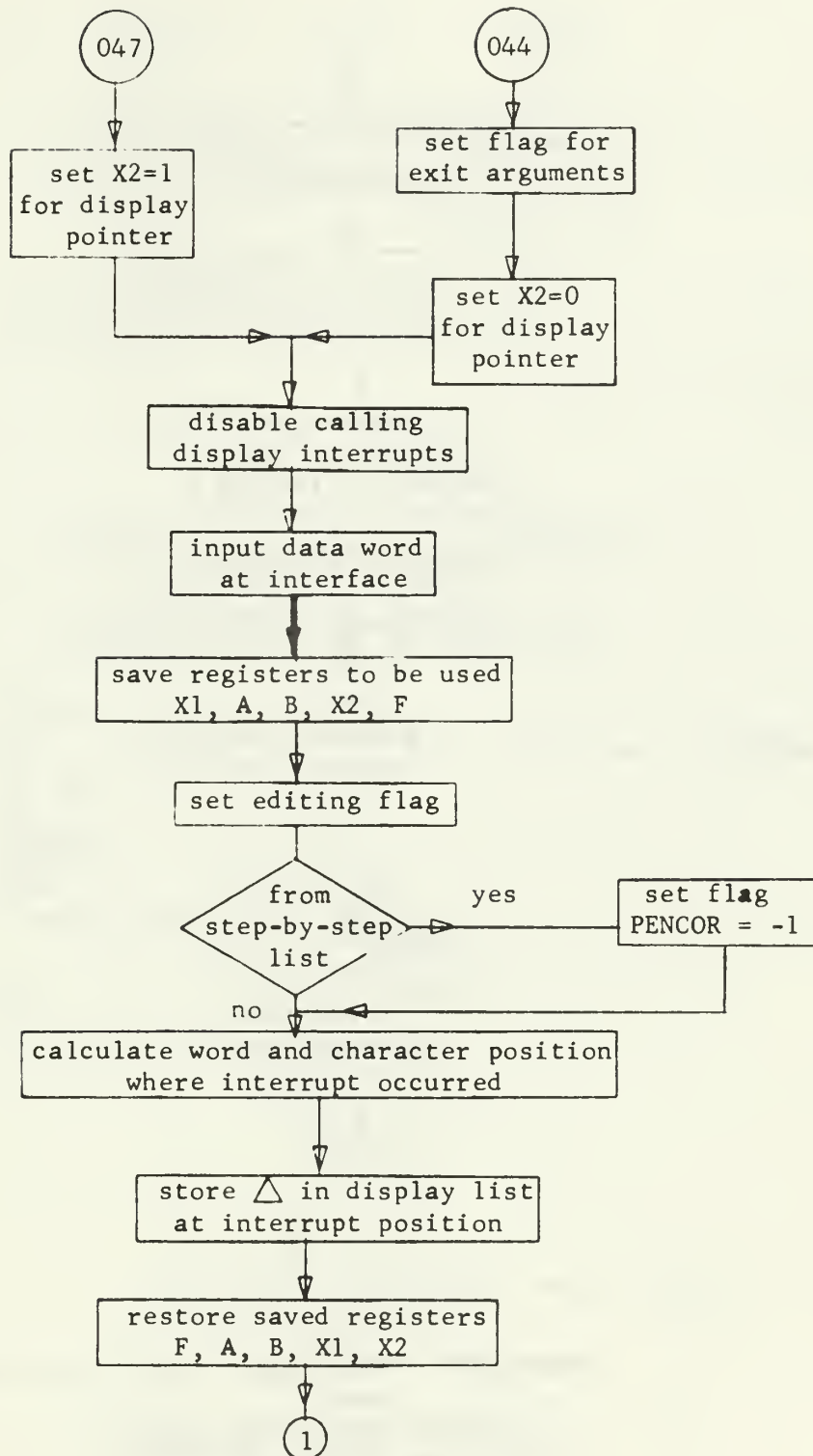
## 10. PEN1ACT and PEN2ACT

These are entered from the light pen interrupts at the displays and set index register 2 (X2) to point to the proper arguments in the re-entrant argument tables. The registers to be used are saved for pre-exit restoring. The display buffer word and character position pointer are read in and processed. The character where the light pen strike originated is replaced with a  $\Delta$ . The function panel and keyboard are enabled for input of the desired correction character. The saved registers are restored and control is returned to the interrupted subroutine.

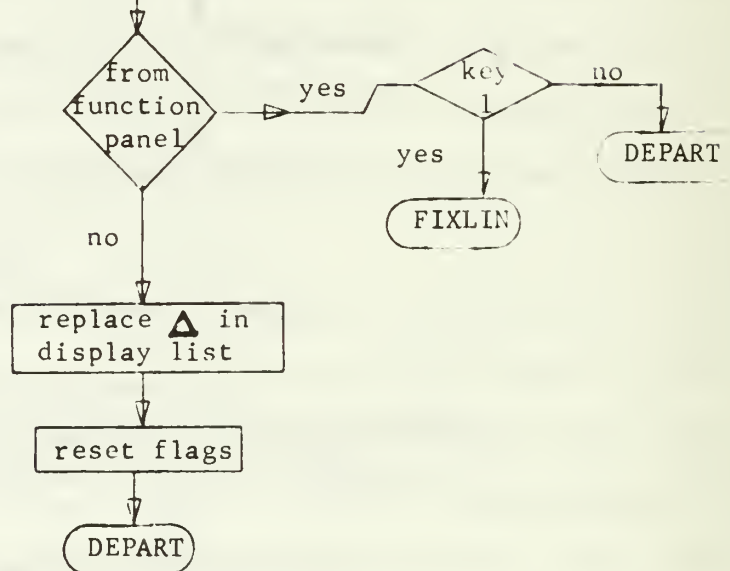
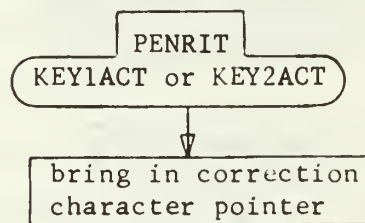
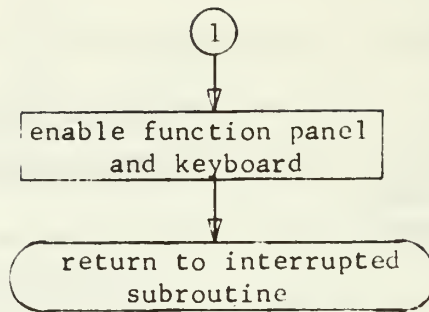
PENRIT is internal to this subroutine and is entered from KEY1ACT or KEY2ACT. It reads the input character and if it is function panel key 1, branches to the subroutine that erases the line where the interrupt occurred. If it is a keyboard character, it is input as a single correction character to replace the  $\Delta$  input at the strike.

## 11. KEY1ACT and KEY2ACT

These are entered from interrupt 045 and 050 respectively. Each interrupt is initiated by a key stroke on the function panel or keyboard of its respective scope. The originating scope is noted, and indexing to the proper argument in the re-entrant argument tables is used. A code check is made to route the input character to the light pen correction routine, erase line and fill action routine, carriage return routine, backspace with erase routine, foreground to background transfer, transfer from input to editing mode or editing to input mode, or the function panel key subroutine address table. These subroutine activities are all monitored, and exit is provided which restores all saved registers. When the page buffer is full the page is

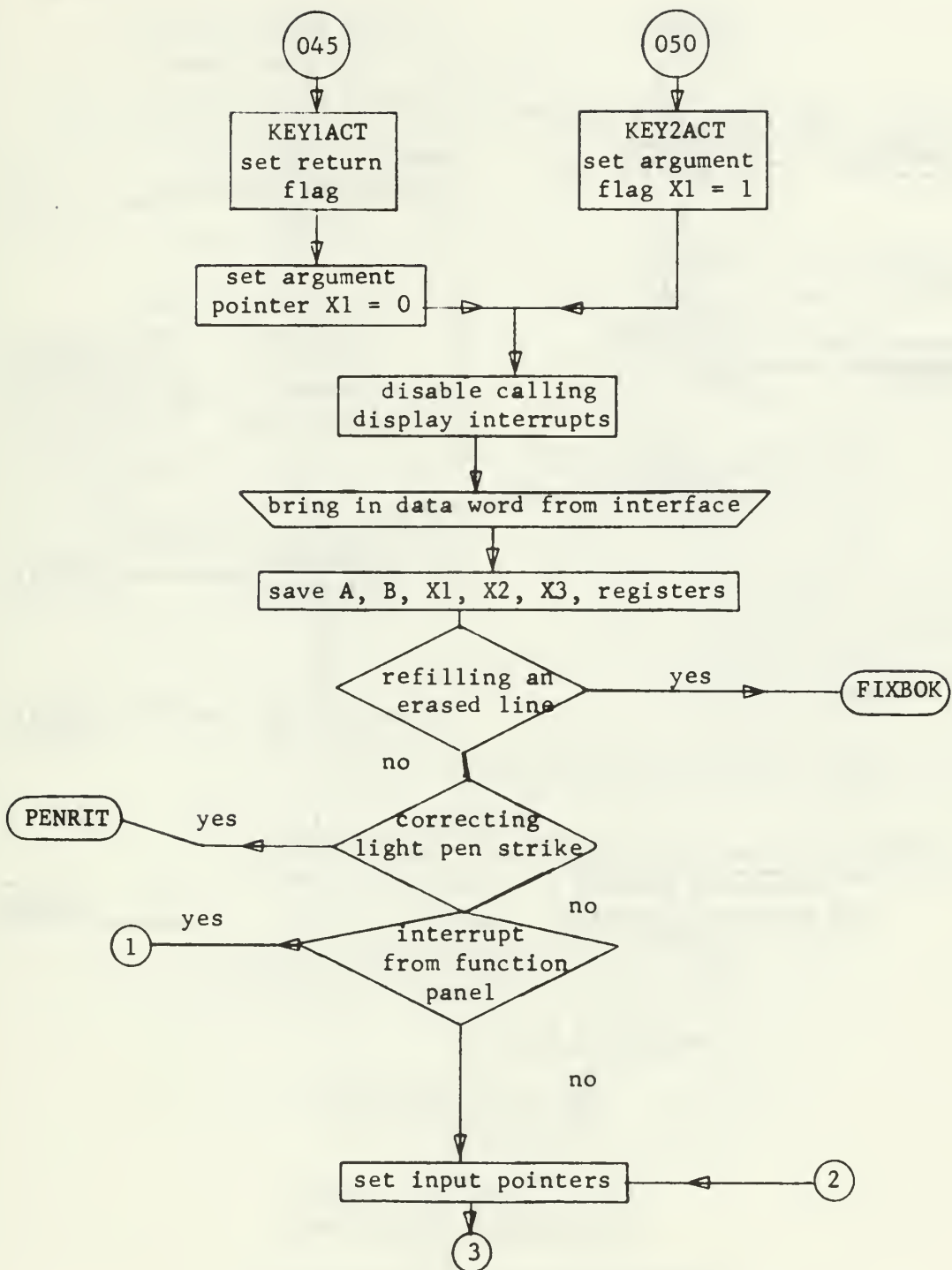


SUBROUTINE PEN1ACT and PEN2ACT

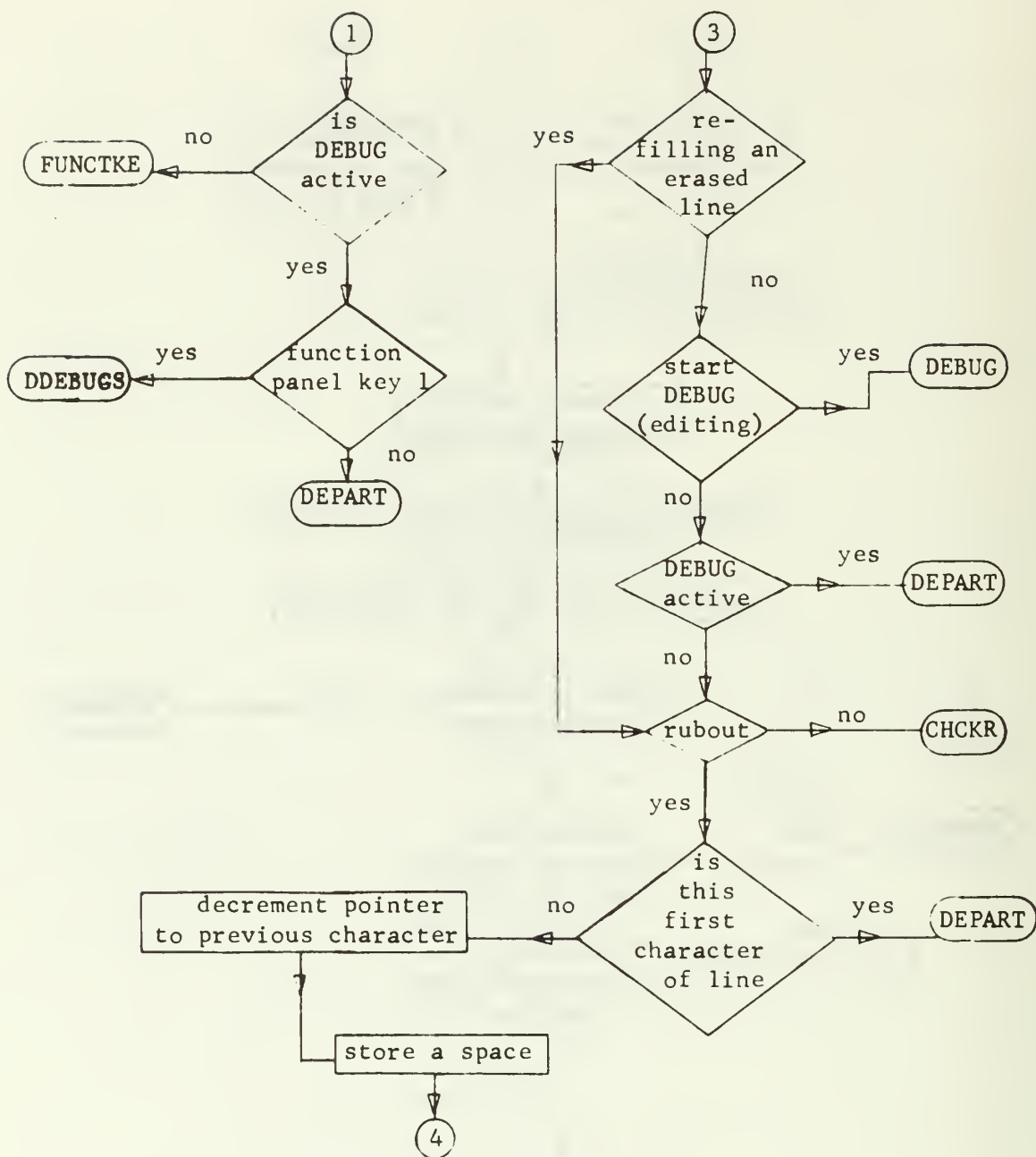


SUBROUTINE PEN1ACT and PEN2ACT (continued)  
and  
INTERNAL SUBROUTINE PENRIT

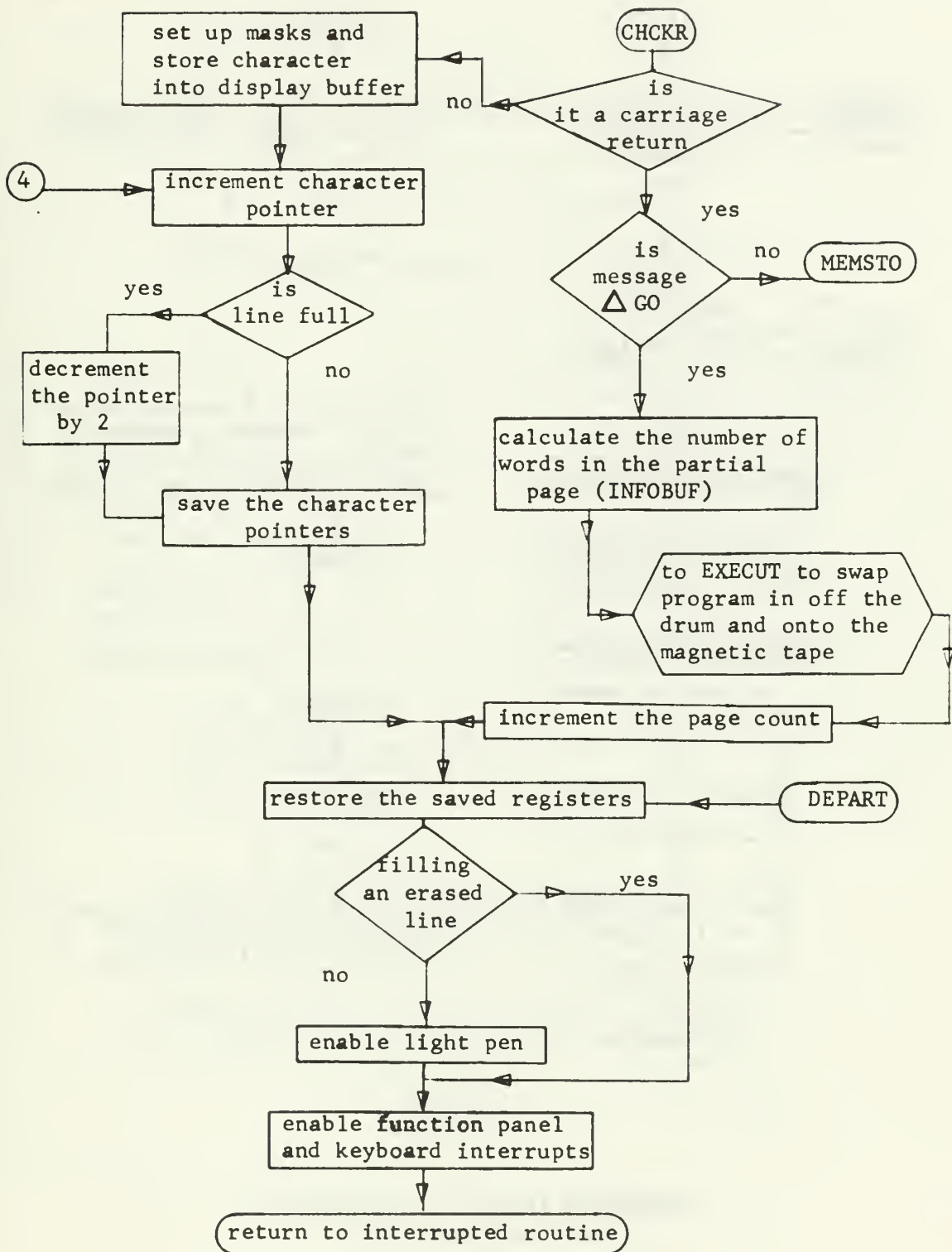




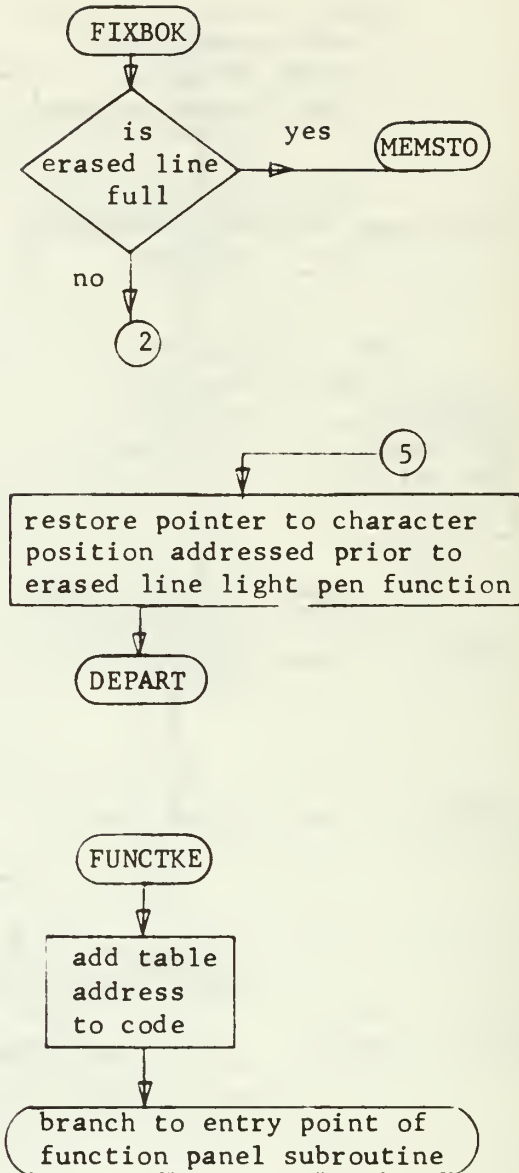
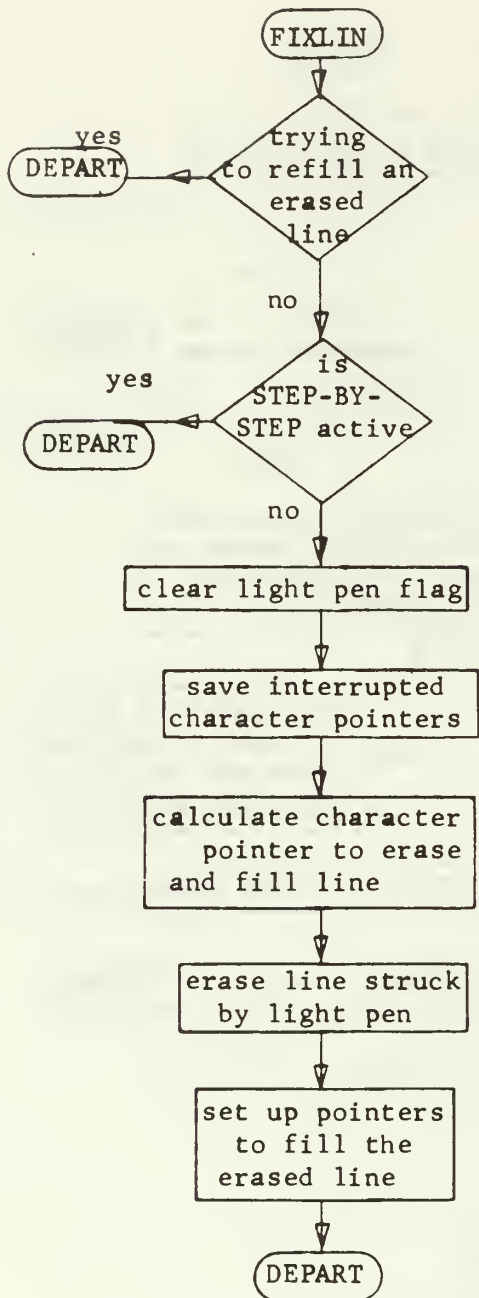
SUBROUTINE KEY1ACT and KEY2ACT



SUBROUTINE KEY1ACT and KEY2ACT  
(continued)

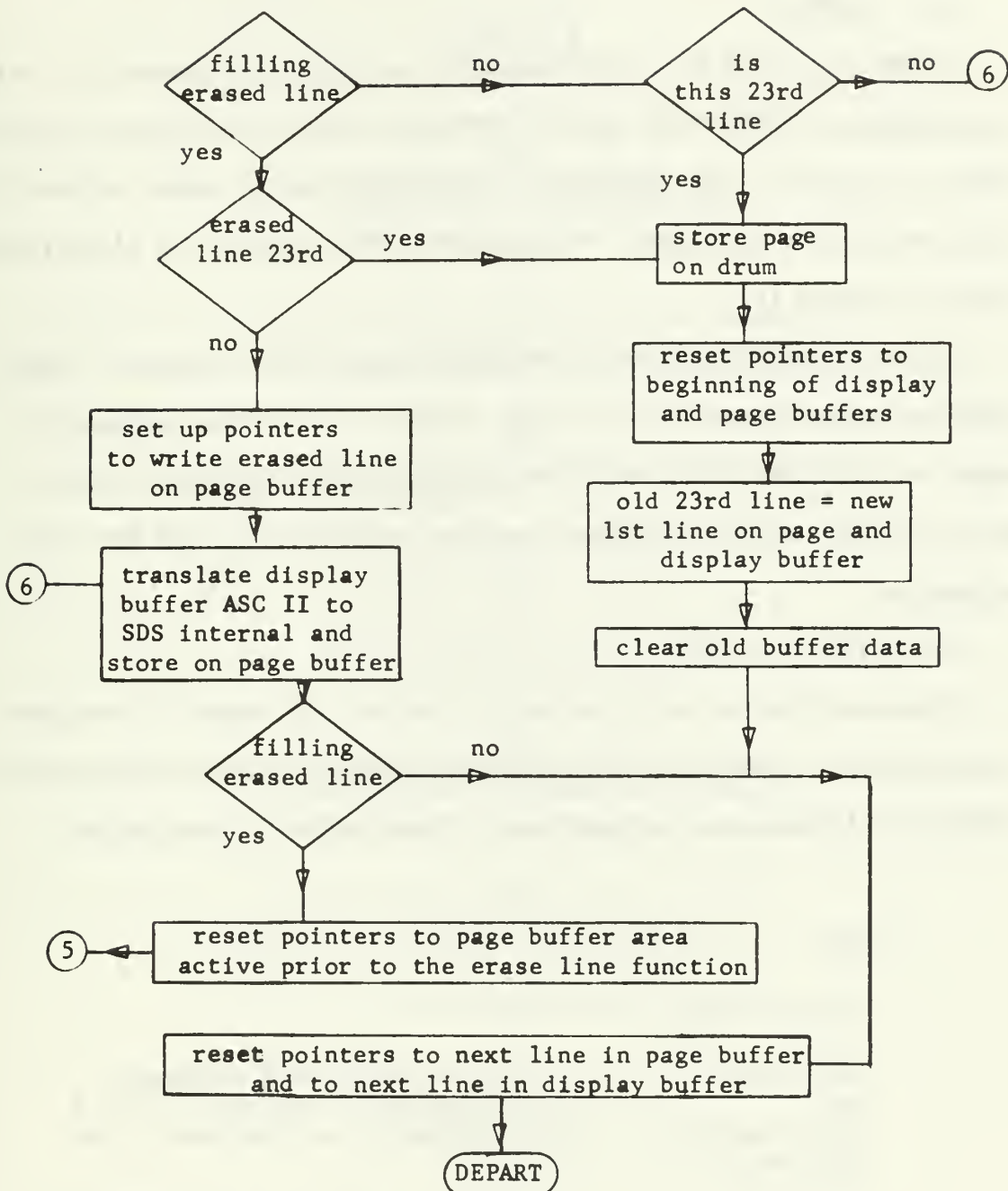


SUBROUTINE KEY1ACT and KEY2ACT  
(continued)



SUBROUTINE KEY1ACT and KEY2ACT  
(continued)

MEMSTO



SUBROUTINE KEY1ACT and KEY2ACT  
(continued)



automatically stored on the drum and pointers are set for inputting the first line of a new page.

## 12. STEPBY

STEPBY is called by a Meta Symbol or Fortran IV program with a calling sequence of BRM STEPBY and PZE NUM where NUM is the number of arguments in the list. The arguments include the display number and up to six variables to be traced. The proper calling sequence is illustrated below in Figure 16.

This call must be the first executable step in the program. STEPBY uses the return address as the first address in the octal program it swaps out onto the drum, notes the calling display, and saves the A, B, X1, X2, X3 and flag registers and the original values of the trace arguments.

## 13. STEP1 and STEP2

These are the two entry points for the main processor in the step-by-step mode. Internal to this are the NOEXU (skip ahead one) and the BACKUP1 (skip backward one) options. These options respectively

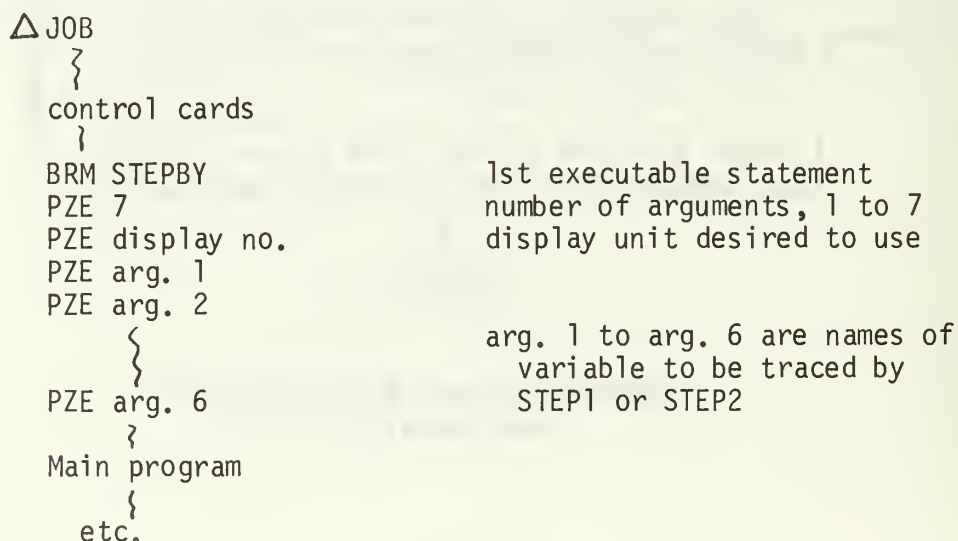
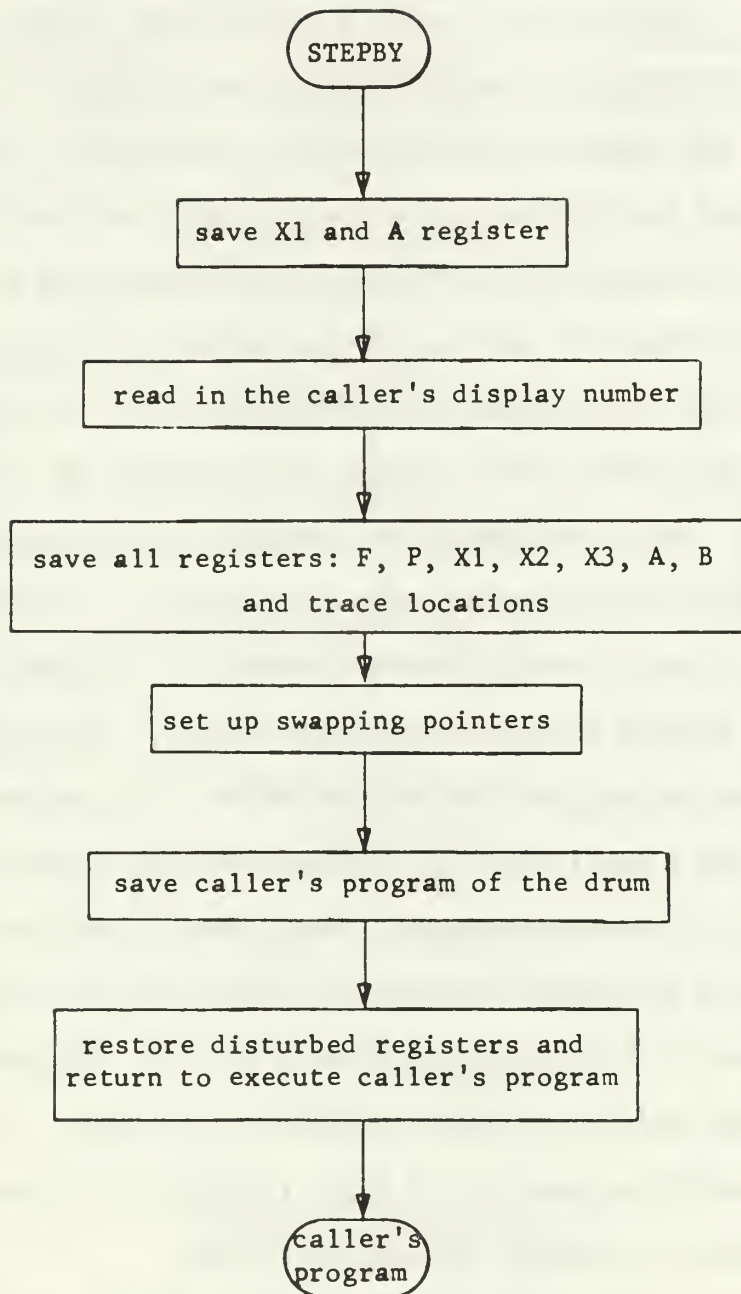


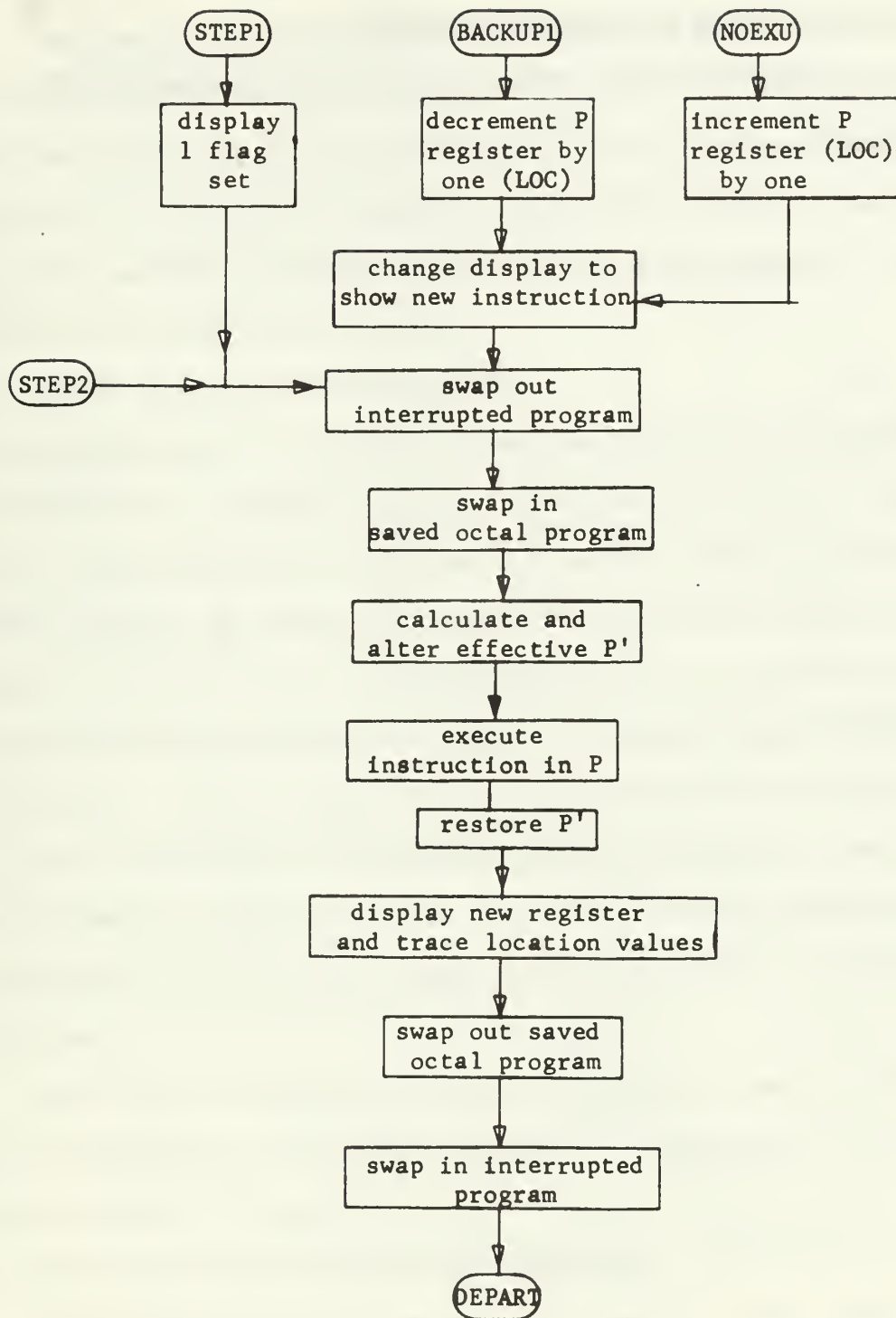
Figure 16



SUBROUTINE STEPBY

increment or decrement the P register by one. Then the instruction corresponding to the P register is displayed in the display instruction register. The light pen is enabled so light pen changes to the displayed X1, X2, X3, A, B, and I registers may be made. I register changes do not change the actual program instructions. The I register may have instruction put in at the console and executed without affecting the contents of the address indicated by the P register. If program changes are desired the appropriate instruction must be entered in the I register. During execution STEP1 or STEP2 swaps in the saved (by STEPBY) octal program, and executes the instruction in location P. Before this execution the action of the execution is determined and if necessary the next P value (P') is calculated. A transfer of control instruction is stored in P' and when P is executed P' returns control to the step-by-step routine. The step-by-step routine returns the original instruction of P'. This sequence is only necessary for a small class of instructions that directly change the P register, for example a branch. Most instructions are executable by a standard Meta Symbol instruction (EXU) that will execute remote instructions by using the instruction address as its operand. The step-by-step routine maintains control in this manner. The results of the execution as seen in the other registers and traced variables are translated to display code and displayed.

All of these subroutines are re-entrant and the re-entrancy is expandable to as many display consoles as there is room in core for their display buffers and all the associated arguments. For each additional console all argument lists would have one corresponding argument added to them. The index register used for pointing will



SUBROUTINE STEP1 and STEP2

point to the argument and is set up the same way the second display is set up for entering the interrupt routines.

For further detail the programs are included in Appendix III.



## VI TIME SHARING SYSTEM EMPLOYMENT

The special purpose time sharing programs are best understood when an information and execution flow is outlined for the computing system when operating under the mutual control of RTM and the time sharing package. The new modes of operation are outlined below in graphical form with amplifying comments to describe the proper sequence of execution and responses from system.

### A. PROGRAM INPUT, EDIT AND DEBUG MODE

Input programs can originate at the card reader or the display console keyboard. The input is always in SDS internal code and resides in the page buffer and on the drum in this form. During editing, DEBUGS controls the paging and translation between the display and page buffers. See Figure 17. The correction activity is controlled by the light pen strikes (PEN1ACT and PEN2ACT) and keyboard inputs (KEY1ACT and KEY2ACT). Once a program is in the page buffer and on the drum it may be transferred to the magnetic tapes and executed in the background queue as often as desired. This allows editing between runs without read in of the whole program at the card reader or keyboard each time.

During input, editing, and transfer to magnetic tape each scope runs independently of the other and both run independent of batch or hybrid programs in the system.

### B. STEP-BY-STEP EXECUTION AND DEBUGGING MODE

Programs may originate from any input console in the system if they conform to the proper calling sequence indicated above in Figure 12. The stepping function is called from KEY1ACT or KEY2ACT by use of function panel keys 29, 30, or 31. STEP1 and STEP 2 control the

SYSTEM DATA FLOW UNDER INPUT, EDITING AND DEBUG MODE

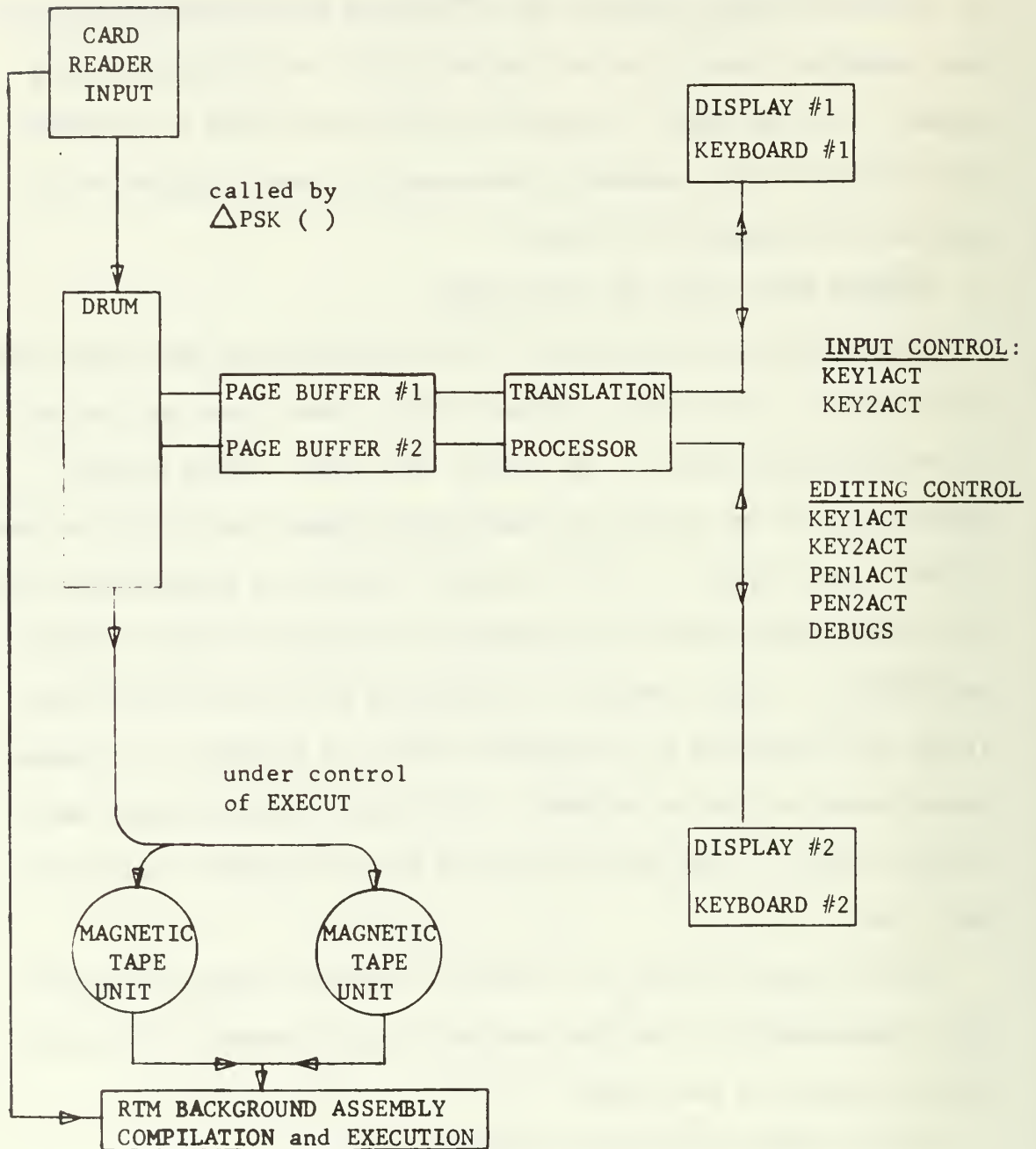


FIGURE 17

# SYSTEM DATA FLOW UNDER STEP-BY-STEP EXECUTION MODE

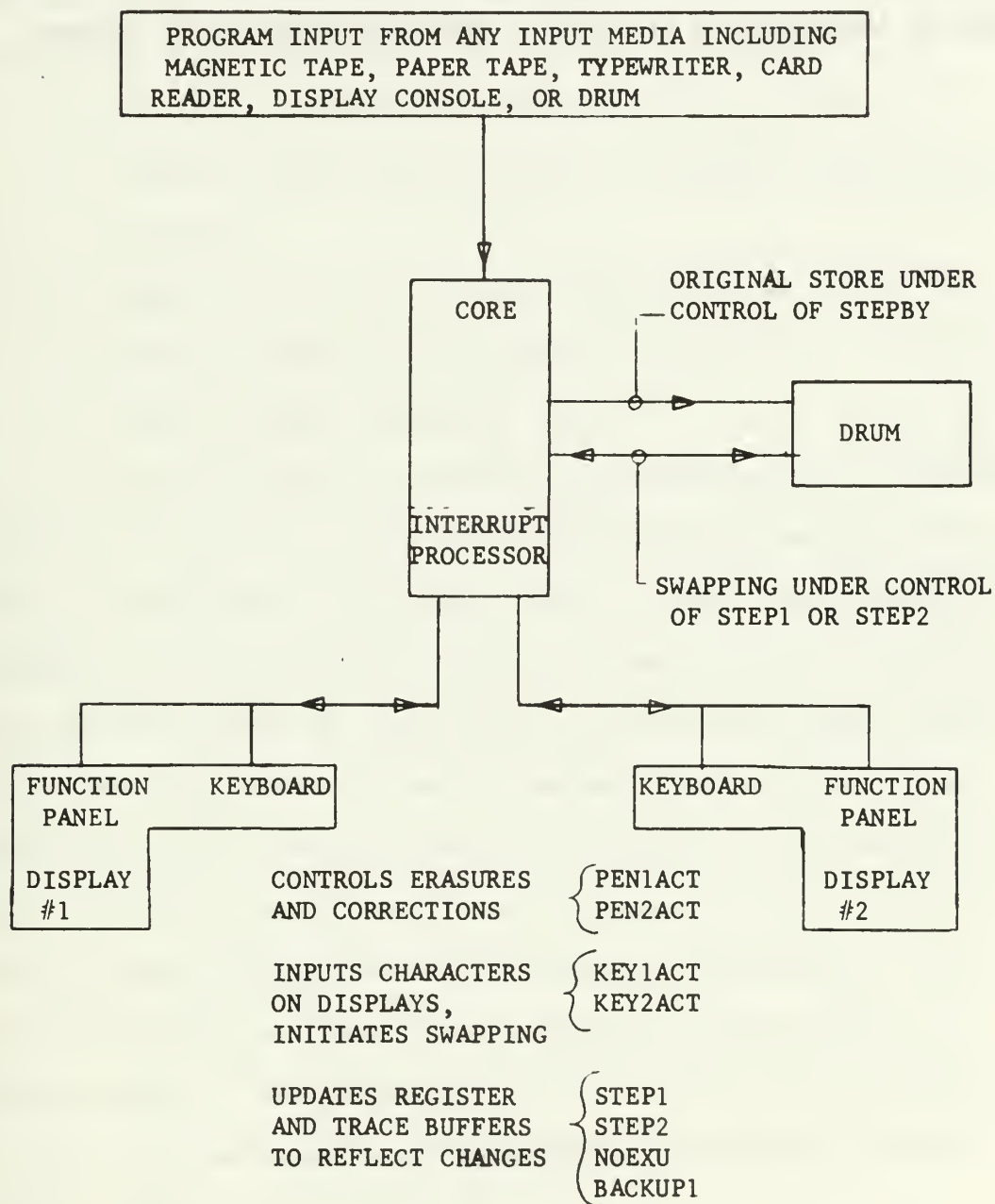


FIGURE 18

swapping, execution of a single instruction and updating of the saved program, if the display buffers are altered by a light pen erasure and keyboard correction. See Figure 18. During the execution of step-by-step tracing at a display each display runs independently of the other and both run independently of batch or hybrid programs in the system.

## VII. CONCLUSIONS

The resultant time sharing system is essentially a special purpose, task oriented time shared capability. The computing system under the combined control of the time sharing package and RTM will provide parallel service, rather than serial, for these principal uses:

1. Step-by-step execution and program activity tracing.
2. Iterative syntax error correction, assembly, compilation, and execution.
3. Symbolic input and editing of object programs.
4. Non-real time hybrid simulation studies.
5. General digital computation tasks.

The CPU idle time during step-by-step execution was the most restrictive, hence the first to be integrated into the package. Previously, the CPU was in idle mode (versus run mode in an idle loop) between each step of stepping through a troublesome loop in a program and could not service any other users. Shifting this mode of operation to the displays to run as an interrupt service routine clears the system of this idle mode restriction and thus allows the hybrid user to also interrupt CPU activity for service. Another user may use the second display terminal for step-by-step execution or for correcting syntax and logic errors (debugging) in the input and editing mode. If the hybrid user is not using all the I/O devices a fourth user can enter the batch queue and use whatever CPU time and I/O capability is left over.

### A. GENERAL CONCLUSIONS

Analysis of the general time sharing problem after an attempt at a coherent system design impressed the need for certain hardware and



software capabilities, and established their hierarchy of importance. Multiple terminal systems satisfying the four primary requirements discussed below are capable of efficient time sharing, without satisfying these requirements time sharing is possible but not necessarily efficient.

A prime factor is a high speed swapping capability which requires a high transfer rate bulk memory device and a very efficient software transfer controller.

A completely interruptable system supervisor with re-entrancy provided for all routines where multiple interrupt-originated entries are possible is a must for coherent, consistent time sharing. This requirement includes re-entrant assemblers and compilers.

The third primary requirement is for parallelism in performance of as many tasks as possible. Time sharing is often thought of as sharing a CPU's time among several users because of the cost inefficiencies involved in idle CPU time. The really expensive item in any computer is the core memory, so it makes more sense to approach the time sharing problem from the viewpoint of exercising the core memory to its maximum, not necessarily the CPU. This can be done by interlacing I/O operations on high speed data channels with CPU memory accesses. To push the memory, multiple CPU instruction and supporting registers accessing blocks of memory in a parallel sense, through MAMS, is the way to extract more use out of a "limited" memory computer. To control and feed a system of this type the first two system requirements must be met.

A fourth primary requirement is for a program interrupt capability so active programs can be interrupted in favor of a higher

priority request. The priority is determined by several factors and normally shifts from user to user under supervisor control.

The more elementary hardware requirements for multiple user terminals and I/O devices must be satisfied but are not discussed here.

#### B. RECOMMENDATIONS FOR THE SYSTEM STUDIED.

The problems encountered in the software implementation phase of this study indicate improvements needed for efficient time sharing on the subject computing system. The following recommended improvements are within the above outlined general time sharing requirements.

A six second response time for the stepping mode and paging activity is due to the RTM drum I/O processor. This subroutine needs updating to get transfer of data up to the hardware rate, not the software rate. A second alternative is obvious, that is use of an independent drum I/O controller (similar to RADOP listed in Appendix A) that accesses the RTM list reservation table.

The need for a second I/O channel becomes obvious from the discussion of Chapter III, Section 5. The second I/O channel ideally would be used for swapping only, hence only have the drum attached. This would reduce potential I/O stalling of the CPU from 100 percent to less than 30 percent during swapping.

The implementation of a more general time sharing capability would require a re-entrant assembly and compilation capability to allow entry from more than one terminal. This can be done with a pseudo re-entrant capability, if desired.

#### C. EXTENSIONS OF THE SPECIAL PURPOSE PACKAGE

The function panel has many open keys available for connection to additional subroutines. Some suggested routines that would be

extensions of the techniques learned in this study and could use some of the subroutines are here proposed.

An adaption of the step-by-step mode at the display consoles to on-line programming techniques where a routine is swapped in, executed, then swapped out is desirable. The results are displayed along with data groups to be varied. The on-line programmer can then change the contents of the data group for a rerun, and start the cycle again. This could be connected to the graphing routine and used in much the same manner except that the on-line programmer would then have graphical results for evaluation. The graphical display could be an optional memory type where useful graphs could be saved and unnecessary graphs rejected. Another linkage could be provided where one scope could be used to drive the other using the graphical and on-line technique outlined above. These functions can be swapped in and out of memory and hence share the CPU with other users on a whole program swapping basis.

## BIBLIOGRAPHY

- Barnes, George H., et al., "The Illiac IV Computer" Institute of Electrical and Electronics Engineers Transactions on Computers, Vol. C-17, No. 8, August 1968.
- Bell, C. G., "Fundamentals of Time Shared Computers", Computer Design, Vol. 7, P 44-47+, February 1968 and Vol. 8, P. 28-43, March 1968.
- Chapanis, Alphose, Man-Machine Engineering, Wadsworth Publishing Company, Inc., 1965.
- Nickerson, Raymond S., Elkind, Jerome I., and Carbonell, Jaime R., "Human Factors and the Design of Time Sharing Computer Systems", Human Factors, Vol. 10, N. 2, P. 127-133, April 1968.
- Nickerson, Raymond S., Elkind, Jerome I., and Carbonell, Jaime R., "On the Psychological Importance of Time in a Time Sharing System" Human Factors, Vol. 10, N. 2, P. 135-142, April 1968.
- SDS Real-Time Monitor Reference Manual for 900 Series/9300 Computer, Scientific Data Systems, Publication Number 90 1108C, July 1967.
- SDS Real-Time Monitor Technical Manual 900 Series/9300 Computers, Scientific Data Systems, Publication 90 11 09A, April 1968.
- SDS Symbol and Meta-Symbol Reference Manual for 900 Series/9300 Computers, Scientific Data Systems, Publication Number 90 05 06F, August 1967.
- SDS 9300 Computer Reference Manual, Publication Number 90 00 50F, March 1967.
- Technical Manual Rapid Access Data File Models 9367B, 9367C, Scientific Data Systems, Publication Number 90 10 29A, October 1967.
- Wilkes, M. V., "The Design of Multiple Access Computer Systems", The Computer Journal, Vol. 10, N. 1, P 1-9, May 1967.
- Wilkes, M. V. and Needham, R. M., "The Design of Multiple Access Computer Systems: Part 2" The Computer Journal, Vol. 10, N. 4, P. 315-320, February 1968.

## APPENDIX A

### CALCULATIONS AND DATA

Stalling percentages were calculated based on the nominal word access rates to main memory for each peripheral device and compared to the memory access cycle time of the CPU. Calculation was based on the following relationship:

Percent Stalling (%) = Peripheral data word memory access rate/CPU data word memory access rate. Example calculation for drum Stalling: (%) =  $147K/572K = 25.7\%$

Averaging of large amounts of raw timing data was used to reduce the data into useable form for graphing the transfer rate characteristic of the drum. The experimental procedure was to time a continuous swapping sequence through 100 operations. The swapping rate was based on a read onto the drum and write back into core operations.

Sample calculation for word swap rate:

$07777 = 4095_{10}$  words swapped in and out 100 times: 14.1 sec. (data)  
 $14.1 \text{ sec} - .1 \text{ sec reaction time} = 14.0$   
 $14.0 \text{ sec.} / 100 = .14 \text{ sec. average transfer rate in and out}$   
 $14 \text{ sec.} / 2 = .07 \text{ sec average transfer rate for 1 band (07777 words)}$   
 $4095 \text{ words} / .07 \text{ sec} = 58500 = 58.5 \text{ KHertz word transfer rate.}$   
 $3584 / .035 = 112000 = 112 \text{ KHertz}$   
 $170.5 / 2 = 85.25 \text{ KHz} = \text{average word transfer rate}$



# APPENDIX A (continued)

## DATA

RUN #	RTM I/O (sec.)	RADOP (sec.)	# of WORDS (K =thousands )
1	14.2	7.2	60.0
2	68.0	7.2	280
3	78.7	7.2	400
4	112.	7.2	600
5	151.	14.1	800
6	195.	14.1	1000
7	237.5	14.1	1200
8	277.5	14.1	1400
9	317.	14.1	1638
10	-	12.1	1800

```

*****  

**  

** RADOP SWAPS A LIST ONTO THE RAD, THEN SWAPS IT OFF INTO CORE.  

** SWAPPING ITERATIONS ARE CONTROLLED BY THE X2 REG.  

**  

$RADOP *****  

NOP =0177634,2  

LDA =01234  

STA DAT  

ARM LDI  

DZE O  

*****  

**  

** HALT TO ALLOW STARTING OF MANUAL TIMER.  

**  

*****  

**  

** SKIP IF DRUM READY FOR OUTPUT FROM CORE  

** NOT READY, WAIT  

** READY, ALERT CHANNEL A FOR POT OF RAD ADDRESS.  

** POT RAD ADDRESS TO CHANNEL CONTROLLER  

** ALERT INTERLACE, I/O MODE  

** I/O IOSD MODE, ZWC INTERRUPT  

** POT WORD COUNT AND ADDRESS OF DATA  

** CONNECT INFO AND RAD FOR INTO RAD  

** SKIP IF CONTROLLER READY  

** NOT READY, WAIT  

** READY, SKIP IF NO DRUM CONTROLLER ERROR.  

** ERROR, TRY AGAIN  

** NO ERROR, SKIP IF NO CHANNEL ERROR  

** ERROR, TRY AGAIN  

** NO ERROR, CONTINUE  

*****  

**  

** SKIP IF CONTROLLOR READY  

** NOT REZDY, WAIT  

** ALERT CONTROLLER FOR POT OF RAD ADDRESS  

** POT SECTOR, BAND, AND UNIT ADDRESS WORD  

** ALERT INTERLACE, I/O MODE  

** I/O IOSD MODE, ZWC INTERRUPT  

** CORE STORAGE AREA ADDRESS AND WORD COUNT  

** READ SECTOR INTO FILE  

** SKIP IF CONTROLLER READY  

** NOT READY, WAIT  

** READY, SKIP IF NO DRUM CONTROLLER ERROR.  

** ERROR, TRY AGAIN  

** NO ERROR, SKIP IF NO CHANNEL ERROR  

** ERROR, TRY AGAIN  

*****

```

```

LDI
DAT
LIST
INFO
WCAD
POTA
RADLCC
WRDLCC
WCADDR

BRX
PZE
LDX
LDA
STA
BRX
RRR
RES
EQU
RES
FORM
FORM
POTA
RADLCC
WCAD
WCAD
END

GOO,2
0
0
=0170000,1
DAT
070000,1
$-1,1
LDI
1
020000
020000
10,14
9,2,7,6
0,0,0117,0
01400,LIST
01400,INFN
RADDP
SWAPPING COMPLETED, STOP MANUAL TIMER.

```

## APPENDIX B

The following table lists the ASC II code used in the display, the SDS Internal code and the corresponding symbol. All numbers are octal.

SYMBOL	ASC II	SDS INTERNAL
0	60	00
1	61	01
2	62	02
3	63	03
4	64	04
5	65	05
6	66	06
7	67	07
8	70	10
9	71	11
SPACE	40	12
=	75	13
HYPHEN	47	14
:	72	15
>	76	16
CHECK	47	17
+	53	20
A	1	21
B	2	22
C	3	23
D	4	24
E	5	25
F	6	26
G	7	27
/	57	61
S	23	62
T	24	63
U	25	64
V	26	65
W	27	66
X	30	67
Y	31	70
Z	32	71
?	77	72
,	54	73
(	50	74
#	43	75
\	34	76
?	77	77
H	10	30
I	11	31
?	77	32

# APPENDIX B (continued)

SYMBOL	ASC II	SDS INTERNAL
.	56	33
)	51	34
[	33	35
<	74	36
End of Line	37	37
-	55	40
J	12	41
K	13	42
L	14	43
M	15	44
N	16	45
O	17	46
P	20	47
Q	21	50
R	22	51
Carriage Return	77	52
\$	44	53
*	52	54
]	35	55
:	56	73
Δ	0	57
Blank	40	60



## APPENDIX C

This Appendix is a complete listing of the subroutines used to operate the special purpose time sharing capability for the studied system. The listings start on the next page.



RRU	DEBUG	SKIP IF DEBUG MODE IS ACTIVE
CKN	RUGGY,1	
RRU	#+2	
RRU	DEPART	MISTAKE, IGNORE INPUT
CKE	RURCUT	SKIP IF PREVIOUS CHARACTER TO BE SCURREN
RRU	CHCKR	
LDA	SPACES	
RRU	*MSG1,1	
COPY	PREVLN	PREVIOUS
CUR	(5,4),(3,5)	A INTO R, X3 INTO A
COPY	=2	A INTO X3, R INTO A
LDR	(5,3),(4,5)	
STS	STSMCK+5,3	
WPN	*MSG1,1	
RRU	TEMPCO,1	
COPY	NXTWRD	UPDATE ERASED LINE FILLER CHARACTER COUNT
CKE	(5,4),(2,5)	
RRU	LINECO,1	IGNORE IF BEGINNING OF LINE
MPN	#+3	
CUR	TEMPCO,1	UPDATE ERASED LINE FILLER CHARACTER COUNT
COPY	DEPART-5	ERRPR, IGNORE INPUT ERASE
LDX	ONE	
RRU	(5,2),(4,5)	
CKE	=0177776,3	SKIP IF INPUT IS CARRIAGE RETURN
LDA	RUBSTO	PUT BEGINNING OF LINE POINTER IN X2
RRU	STCWRD	
COPY	LINECO,1	
CKE	(5,2)	
RRU	*MSG1,1	
COPY	DELTO	SKIP IF LINE READS -GO
MPN	MEMSTO	
COPY	(0,4)	CALCULATE
COPY	077777,(2,4)	THE NUMBER OF WORDS
LLSR	(0,5)	ON THE PARTIAL
DIV	01	PAGE AND STORE IN WORD BUFFFFE
MUL	TWENTY	OF EXECUT
LLSR	23	
STA	INFCRUF,1	
LDA	PRUMST	ADD PARTIAL PAGE TO
RRM	SVDRUM	END PROGRAM LISTING.
LDA	PRUMST	
STA	SVDRUM	
RRM	PRUMST	
MPN	EXECUT	
RRU	FACES,1	
	DEBAR	

STWARD	STC	MSG1,1	\$+16,3	STCMSGK+3
	STC	\$+1,2	\$+1,2	SHIFTL+4,3
NXTWRD	BRX	(0,5)	(0,5)	STCMSGK+4,3
	CCPY	(2,4)	(2,4)	
	CCPY	034010	034010	
	RCH	1	1	
	LISB	TWNS	TWNS	
	DIV	(4,5)	(4,5)	
	COPY	=0	=0	
	SKN	\$+6	\$+6	
	ARU	(2,5)	(2,5)	
	COPY	ONE	ONE	
	SUR	(5,2)	(5,2)	
	COPY	=0177777,3	=0177777,3	
	LDX	\$+2	\$+2	
	LDX	RESTART,3	RESTART,3	
	CCPY	(2,4)	(2,4)	
	STR	SAVDX2,1	SAVDX2,1	
	COPY	(3,4)	(3,4)	
	STR	SAVDX3,1	SAVDX3,1	
\$DEPART	LDX	SAVFB,1	SAVFB,1	
	COPY	SAVEX2,1	SAVEX2,1	
	LDX	(5,2)	(5,2)	
	COPY	SAVEX3,1	SAVEX3,1	
	LDX	(5,3)	(5,3)	
	COPY	SAVFA,1	SAVFA,1	
	LDX	FIXING,1	FIXING,1	
	SKN	\$+2	\$+2	
	ARU	\$+3	\$+3	
	ARU	ENRPN,1	ENRPN,1	
	EXU	ENBFCN,1	ENBFCN,1	
	EXU	ENBKEY,1	ENBKEY,1	
	EXU	0101	0101	
	FSTP	\$+3	\$+3	
	ARU	SVX102,1	SVX102,1	
	MX	*KEY2ACT	*KEY2ACT	
	BRC	SVX101,1	SVX101,1	
	MX	*KEY1ACT	*KEY1ACT	
	BRC	FIXING,1	FIXING,1	
\$FIXLIN	SKN	\$+2	\$+2	
	ARU	DEPART	DEPART	
	ARU	ISHT,1	ISHT,1	
	SKN	\$+2	\$+2	
	ARU	DEPART	DEPART	

STORE CHARACTER  
IN DISPLAY  
BUFFER.

SETUP INDEX REGISTERS FOR NEXT CHARACTER  
CHECK  
TO INSURE THAT  
CLEAR RITS / 0-8 INCLUSIVE IN R

NO MORE  
THAN  
TWENTY WORDS  
ARE WRITTEN  
PER LINE  
OF TWENTY ARE IGNORED

SAVE  
ROUTINE  
CHAPACTER POSITION CONTROL  
REGISTER  
RESTORE  
REGISTER  
AND  
ENABLE  
INTERRUPTS  
SKIP IF REFILLING ERASED LINE

THEN  
TO  
RETURN  
TO  
INTERRUPTED  
ROUTINE.

SKIP IF TRYING TO REFILL AN ERASED LINE

SKIP IF STEP BY STEP PROGRAM ACTIVE.

LDA	MSG1,1	CALLING DISPLAY LIST ADDRESS INTO A
SKN	EDITING,1	UP DATE EDITING (L.P. FLAG) IF REQUIRED.
ARU	\$+2	
RPT	EDITING,1	RE-SET EDITING FLAG
RCH	034001	CLEAR BITS 0-8 INCLUSIVE
STA	TEMPRAS,1	
LDA	LINFCO,1	SAVE INPUTTING X2 POINTER
STA	TEMPLI,1	FOR LATER RECOVERY
LDA	PENWRD,1	
CUR Y	TEMPRAS,1	SUBTRACT STRIKE LOCATION TO GET RELATIVE
COPY	(0,4)	POSITION OF WORD IN DISPLAY LIST.
LRSC	23	
COPY	(0,5)	
DIV	TWEN5	INTEGER PORTION OF LINE NUMBER IN A
COPY	(0,4)	WORD LOCATION IN LIST INTO R NOT NEEDED
STA	LISTNG,1	SAVE INTERRUPTED INPUTTING LIST ADDRESS
MUL	TWEN5	ADDRESS OF FIRST WORD IN LINE TO ERASE
LSN	23	INTO THE A REGISTER
ADD	FIVE	
RCH	034001	CLEAR BITS 0-8 INCLUSIVE IN A
ADD	=010000	MERGE BRANCH INCREMENT OF ONE
COPY	(5,2)	RELATIVE WORD POINTER INTO INDEX 2
STA	LINFCO,1	
LDX	=0177754,3	X3=WORD COUNTER
LDA	SPACES	STORE SPACES
STX	*MSG1,1	TO ERASE LINE
BRX	\$+1,2	
BRX	\$-2,3	
COPY	(2,5)	RE-POINT TO BEGINNING OF LINE
SUB	TWENTY	
LCR	SAVDX2,1	SET UP POINTERS FOR ERASED LINE
STA	SAVDX2,1	CORRECTION, SAVING ACTIVE INPUT POINTERS
STR	TEMPX2,1	
LDA	SAVDX3,1	
STA	TEMPX3,1	
LDA	RESTAR,1	
STA	SAVDX3,1	
COPY	(2,5)	
SKU	TEMPLI,1	SET FLAG FOR REFILLING ERASED LINE
ARU	DEPART	
LDA	MINUS1	
STA	FIXING,1	
LDA	=RC	
STA	TEMPCO,1	SET-UP WORD COUNT FOR REFILL CHECKING
BRU	DEPART	FILL-ER UP
SKR	TEMPCO,1	SKIP IF ERASED LINE IS FILLED
ARU	\$+10	
ARU	CHKCR+2	





SAVE X2	DATA	0	
SAVE X3	DATA	0	
SAVE A	DATA	0	
SAVE R	DATA	0	
STSM SK	DATA	0	77000000,0770000,07700,077
SHIFTR	DATA	06021022	,06021014,06021006,01000000
SHIFTL	DATA	06025022	,06025014,06025006,01000000
RUR CUT	DATA	0377	
CR	DATA	0215	
DEL TGO	DATA	00071740	
ALTMCD	DATA	0375	
SPACES	CHAR	32,32,32,32	
CHAR TY	DATA	6,6,5,6	
TWEN5	DATA	20	
ONE	DATA	25	
RESTART	DATA	0177774	
LITNO	DATA	0	
TEMPX1	DATA	0	
TEMPX2	DATA	0	
TEMPX3	DATA	0	
TEMPBAS	DATA	0	
TEMPCO	DATA	0	
MSG1	PZF	0	
	PZE	STAR1,2	
		STAR2,2	
MINUS1	DATA	-1	
MEMSTO	SKN	FIXING,1	
	RRU	\$+16	
	LDA	SWAPPER,1	
	STA	TEMPSW,1	
	LDA	LITNO,1	
	SKE	TWEN2	
	RRU	\$+5	
	LDA	TEMPSW,1	
	STA	SWAPPER,1	
	MPT	FIXING,1	
	RRU	\$+7	
	MUL	TWENTY	
	LLSN	23	
	RCH	034001	
	ADD	SWAP,1	
	STA	SWAPPER,1	
	RRU	\$+23	
	MPO	DISLGH,1	
	LDA	DISLGH,1	
	SKG	TWEN2	
	RRU	\$+10	
	LDA	=0100005	

SKIP IF FILLING FRASED LINE
SAVE INTERRUPTED POINTER TO CORF AND
INSERT POINTER TO ERASED AND NOW
CORRECTED LINE
SKIP IF LIST IS FULL
RESTORE POINTERS AND FINISH NORMALLY
RESET FILL FRASED LINE FLAG
CALCULATE CORF ADDRESS OF FILL LINE
AND STORE IT IN CORE
POINTER WORD
INCREMENT LINE COUNTER
CHECK COUNT
SKIP IF INPUT LINE IS EXCESS 23PD LINE

STA	LINECO,1	RE-INITIALIZE ALL POINTERS
STAY	SAVDX2,1	
COPY	(5,2)	
LDA	MINUS1	
STA	CYCLE,1	
LDA	=0177773	
STA	SAVDX3,1	
LDA	DRUMST	
STA	SVDRUM	
ARM	DRUMST	
LDA	SVDRUM	
STA	DRUMST	
COPY	(0,5)	
STA	DISLGH,1	
LDA	SWAP,1	
STA	SWAPPER,1	
MPO	FACES,1	
LDX	=0177754,3	
SKN	CYCLE,1	
ARU	\$+4	
LDA	*MSG2,1	
STA	*MSG1,1	
ARU	\$+2	
LDA	*MSG1,1	
LRSD	18	
ADD	FIRST	
STA	DIRT,1	
LDA	*DIRT,1	
RCH	0331	
LLSD	06	
ADD	FIRST	
STA	DIRT,1	
LDA	*DIRT,1	
RCH	0331	
LLSD	06	
ADD	FIRST	
STA	DIRT,1	
LDA	*DIRT,1	
RCH	0331	
LLSD	06	
AND	FIRST	
STA	DIRT,1	
LDA	*DIRT,1	
RCH	0331	
STR	*SWAPPER,1	
ARX	\$+1,2	
MPO	SWAPPER,1	
SKN	CYCLE,1	

GO TO	SET	MOVE 23RD LIST TO 1ST LIST FLAG
DRUM STORE		
TO STORE 22 LINE PAGE		
OF PROGRAM		
RESTORE INTERRUPTED ADDRESS		
RESTORE LINE COUNTER		
RESTORE LINE TO MEMORY TRANSFER POINTER.		
INCREMENT PAGE COUNTER		
20 WORD COUNT IN X3 REGISTER		
SKIP IF ERASE FILL LINE IS 23RD LINE		
TRANSFER 23RD LINE TO FIRST POSITION		
IN DISPLAY LIST AND CORE STORAGE		
TRANSFORM ASA22 CODE TO SDS INTERNAL CODE		
SKIP IF ERASE FILL LINE IS 23RD LINE		







```

SET STEP-BY FLAG TO UNCALLED.
RESFT CALLING DISPLAY FOR INPUT/OUTPUT.

ENABLE CALLING KEYBOARD.
  FNABLE CALLING FUNCTION PANEL.
  ENABLE CALLING LIGHT PEN.
RESTORE DISTURBED REGISTERS.

```

R88888

STA	FIRSPAS,1	
LDA	CARDLIN,1	
STA	*WHERE,1	
EXU	ENBKEY,1	
EXU	ENBFCN,1	
EXU	ENBPN,1	
LDA	SAVEA	
LDX	SAVEFX1,1	
LDX	SAVEFX2,2	
LDX	SAVEFX3,3	
RRR	//KEYD	
CARDLIN	0,2,NUMS	
CWFM	0,2,SMUN	
FORM	1,8,15	
WHERE	NUMMER1,NUMMER2	
LPLOC	044,047	
LPSEV	PFN1ACT	
KYLCC	PFN2ACT	
KYSER	045,050	
ENDLCC	KEY1ACT	
ENDSER	KEY2ACT	
SAVEA	043,046	
SAVEFX1	FLASH1	
SAVEFX2	FLASH2	
SAVEFX3	0	
DELIM	0	
TABLE	0	
	6,18	
	2	
	1	
	1,1,LINK1	
NOCHAR	0	
STRING	1	
ISHOCT	1,1	
IPSPAS	0,0	
INFOREF	1,1	
PENDING	0,0	
FRACES	0,0	
SVSPAC	0	
SVDFLM	0	
FIXING	1,1	
EXITING	0,0	
RLGGY	1,1	
LINECO	1,1	
SAVXX2	0100005,0100005	
SAVXX3	0100005,0100005	
	01777774,01777774	







```

LDA STA
MPO LDA
LDA SKF
ARR COPY
STA COPY
ARR STA
NOP $DDERUG
LDA STA
ARM STA
LDA STA
LDA STA
LDA STA
LDA STA
STA BRU
MPT COPY
STA COPY
MPT STA
LDA COPY
LDX LDA
MPO LDA
STA STA
ARRX STA
ARRX LDA
STA BRU
SVUNBUG DATA
MINUS1 DATA
SVMULE PZE
MSG1 PZE
SWAPPER DATA
SWAP DATA
EXTRA DATA
DEXTRA DATA
EXTRA1 RES
EXTRA2 $SVDRM
$SYST DATA
SYST

SVUNBUG
UNBUG
ENDING,1
FACFS,1
ENDING,1
MULENT
(O,5)
ENDING,1
MULENT
UNBUG
SVUNBUG
UNBUG
SVUNBUG
UNBUG
FACES,1
ENDING,1
MINUS1
MEMS,1
OLDPIC-5
MEMS,1
(O,5)
ENDING,1
RUGGY,1
LINECO,1
(5,2)
=0177754,3
*EXTRA,1
*MSG1,1
$+1,2
$-4,3
DEXTRA,1
DEXPART
O-1
O-1
STAR1,2
STAR2,2
INFO1,INFO2
INFO1,INFO2
EXTRA1,EXTRA2
EXTRA1,EXTRA2
20
20
STLIST

RAD
KEEP TRACK OF PAGES
SKIP IF LAST PAGE IS ON DISPLAY

STORE
CORRECTED
PAGE
ON
RAD
RESET PAGE NUMBER

SET LOCATION FLAG
ERSET LOCATION FLAG

RESET PAGE POINTER FOR DEBUGGING.
RESET DEBUG ACTIVE FLAG TO INACTIVE
SETUP TO
RESTORE LSAT
ACTIVE LINE ON DISPLAY.

AND DISPLAY

```



```

NUMBUF DATA
DIRTY DATA
MEMS DATA
FIRST DATA
TWEN2 DATA
FIVE DATA
END

```

```

*****
O,C
O,C
1,1
LIST
22
05
*****
PENIAC AND PEN2ACT
*****

```

# LIGHT PEN SERVICE ROUTINE

```

*****
$PENIAC P7F
0
07601
SVX2D1,2
$+4
*****
$PEN2ACT P7F
0
SVX2D2,2
07700
DISABL,2
EOMPEN,2
PENWRD,2
SAVEA,2
SAVER,2
(1,5)
SAVEX1,2
=-1
EDITING,2
ISHT,2
$+2
PENCR,2
PENWRD,2
=-1
22
SA VX1,2
(5,1)
02
NWRD,2
*PENWRD,2
*WPE,1
*PENWRD,2
*SAVEA,2
*SAVER,2
*SAVEX1,2
(5,1)
*****
SET FLAG 5 TO INDICATE ENTRY FROM PEN 1
SET X2=0 FOR DISPLAY 1, SAVE INTERRUPTEDX
*****
PEN 2 ENTRY POINT
SET X2=1 FOR PEN2, SAVE INTERRUPTED X2
RESET ALL FLAGS
DISABLE CALLING DISPLAY INTERRUPTS
ADDRESS CALLING PEN FOR PIN WORD
STORE PINNED PEN WORD
SAVE INTERRUPTED A
SAVE INTERRUPTED A
X1 INTO A
$SAVE X1
*****
STORE LIGHT PEN EDITING FLAG
IF ONE-SHOT ACTIVE, SET ONE-SHOT CORRECT-
ING FLAG.
LD A WITH PINNED PEN WORD
PROCESS PEN WORD
TO PUT
ADDRESS OF STRIKE INR,
CHARACTER POSITION IN X1
*****
ERASE STRUCK
CHARACTER
RESTORE AND
EXIT
*****
RESTORE X1
*****

```

[illegible]

\$>>>PSK  
 LINK1  
 AGAIN  
 UPDATE

PZE  
 FIRM  
 ARM  
 PZE  
 RCH  
 DIR  
 LDA  
 LRSA  
 RCH  
 SUB Y  
 COPY  
 FIR Y  
 CCP Y  
 STA  
 LDX  
 LDA  
 LCB  
 STS  
 LRSA  
 LDB  
 STS  
 CAT  
 BRX  
 BRU  
 CRT  
 RCD  
 EXU  
 PCT  
 CAT  
 BRU  
 LDA  
 ADM  
 LDA  
 SKE  
 ARU  
 ARU  
 LDA  
 STA  
 ARM  
 LDA  
 STA  
 FSTR  
 ARM  
 MPO  
 ESTP

0 07700  
 \ KEYD  
 R \ SCAN  
 1 TABLE  
 STRING  
 12  
 037601  
 =01  
 (5,1)  
 (0,5)  
 FACES,1  
 =02477054,2  
 DINF,1  
 =037777  
 WCADDR,1  
 0  
 =040  
 HIADD,1  
 0  
 \$-1  
 \$+2,2  
 UPDATE  
 0,1  
 \$-1  
 \$0,1,4  
 HIADD,1  
 WCADDR,1  
 0  
 \$-1  
 =20  
 WCADDR,1  
 #DINFO,1  
 ECF  
 \$-13  
 EXE  
 DRUMST  
 SVDRUM  
 DRUMST  
 SVDRUM  
 DRUMST  
 04  
 EXECUT  
 FACES,1  
 04

RESET ALL FLAGS  
 START UP THE DISPLAY

DISABLE ALL INTERRUPTS

CLEAR ALL BUT THE LAST 3 BITS OF A.

RESET PAGE COUNT TO 0 FOR NEW INPUT.  
 SET UP FOR CARD STORING  
 STARTING ADDRESS OF BUFFER  
 MASK IN R FOR LOW ADDRESS STORE IN POT  
 LOWER BITS OF BUFFER ADDRESS INTO POT  
 SET UP ADDRESS FOR HI BIT STARE IN EOM  
 MASK FOR HI ADDRESS STORE  
 HIGH ADDRESS BIT IN EOM  
 SKIP IF CHANNEL A NOT ACTIVE.

SKIP IF CARD READER IS READY FOR INPUT.  
 READ CARD INTO CHANNEL A, 4 CHARS PER WD  
 EOM I/O MODE AND HI ADDRESS BIT  
 WORD COUNT AND LOWER ADDRESS BITS OF POTD

INCREMENT INPUT STORAGE BUFFER ADDRESS  
 LATEST CARD IMAGE INTO A  
 SKIP IF CARD IS A ~EQF



GRIND

EXU	RFW,1	REWIND TAPE 3 OR 4 FOR DISPLAY NUMREP
EXU	TRT,1	1 OR 2 RESPECTIVELY IN PREPARATION
CAT	0	FOR WRITING CALLERS PROGRAM ON TAPE
BRU	\$-2	IN PREPARATION FOR EVENTUAL EXECUTION.
BRU	RTT,1	SKIP IF REWIND NOT SUCCESSFUL.
BRU	\$+2	
BRU	GRIND	
EXU	EFT,1	
ECM	014000	
PCT	SPACE	
EXU	TRT,1	
CAT	0	
BRU	\$-2	
EXU	\$+1,1	ERASE A LEADER ON THE TAPE.
BRU	\$+32	
BRM	R\IOPS	
P7E	1	
OPFDT	0,032,FDT\$2R	TAPE READY, BRING IN CALLERS PROGRAM
LDA	FDT\$2R	FROM THE RAD AND TRANSFER IT TO MAG TAPE.
SKA	RITO	REWIND
BRU	\$-2	TO THE BEGINNING OF
BRM	R\IOPS	SAVE CORE LISTING ON THE RAD.
P7E	1	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
OPFDT	0,040,FDT\$2W	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
LDA	FDT\$2W	READ
SKA	RITO	CONTENTS OF CORE
BRU	\$-2	TO BE SAVED ONTO THE RAD.
BRM	R\IOPS	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
P7E	1	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
OPFDT	0,031,FDT\$2R	WRITE
LDA	FDT\$2R	FND OF FILE MARK ON RAD
SKA	RITO	AT END OF TRANSMISSION.
BRU	\$-2	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
BRM	R\IOPS	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
P7E	1	REWIND
OPFDT	0,032,FDT\$2P	TO THE BEGINNING OF
LDA	FDT\$2P	THE CALLERS PROGRAM ON THE RAD.
SKA	RITO	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
BRU	\$-2	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
BRM	R\IOPS	WRITE
P7E	1	THE CALLERS PROGRAM
OPFDT	0,00,FDT\$2C	INTO CORE FROM THE RAD.
LDA	FDT\$2C	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
SKA	RITO	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
BRU	\$-1	
BRM	\$+31	REWIND
BRM	R\IOPS	TO THE BEGINNING OF
P7E	1	



OPFDT	0,032,FDT51R	SAVE CORE LISTING ON THE RAD.
LDA	FDT51R	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
SKA	RIT0	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
BRU	\$-2	
BRM	R/10PS	
PZE	1	READ
OPFDT	0,040,FDT51W	CONTENTS OF CORE
LDA	FDT51W	TO BE SAVED ONTO THE RAD.
SKA	RIT0	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
BRU	\$-2	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
BRM	R/10PS	
PZE	1	WRITE
OPFDT	0,031,FDT51R	END OF FILE MARK ON RAD
LDA	FDT51R	AT END OF TRANSMISSION.
SKA	RIT0	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
BRU	\$-2	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
BRM	R/10PS	
PZE	1	REWIND
OPFDT	0,032,FDT51R	TO THE BEGINNING OF
LDA	FDT51R	THE CALLERS PROGRAM ON THE RAD.
SKA	RIT0	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
BRU	\$-2	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
BRM	R/10PS	
PZE	1	WRITE
OPFDT	0,00,FDT51C	THE CALLERS PROGRAM
LDA	FDT51C	INTO CORE FROM THE RAD.
SKA	RIT0	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
BRU	\$-2	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
BRM	WTD,1	
EXU	014000	
ECM	WCADDR,1	WRITE PROGRAM
POT	0	ON MAG TAPE.
CAT	\$-1	
ARU	=20	
LDA	WCADDR,1	
ADM	\$-7	
BRX	=01216000	
LDA	WCADDR,1	
STA	\$+1,1	
EXU	\$+14	
ARU	R/10PS	
BRM	1	RESTORE THE CONTENTS OF COEE.
PZE	0,032,FDT52R	
OPFDT	FDT52R	REWIND
LDA	RIT0	TO THE BEGINNING OF
SKA	\$-2	SAVE CORE LISTING ON THE RAD.
BRU	R/10PS	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
BRM	1	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
PZE		WRITE
		THE SAVED CONTENTS OF CORE

CPENT  
LDA  
SKA  
BRU  
BRU  
BRU

PRESENT

OPEDT  
LDA  
SKA  
BRU

WTFEND, I  
WTFEND, I  
WTFEND, I

11  
2  
1  
0

INFC0	INFC1	INFC2	INFC3	INFC4	INFC5	INFC6	INFC7	INFC8	INFC9	INFC10	INFC11	INFC12	INFC13	INFC14	INFC15	INFC16	INFC17	INFC18	INFC19	INFC20	INFC21	INFC22	INFC23	INFC24	INFC25	INFC26	INFC27	INFC28	INFC29	INFC30	INFC31	INFC32	INFC33	INFC34	INFC35	INFC36	INFC37	INFC38	INFC39	INFC40	INFC41	INFC42	INFC43	INFC44	INFC45	INFC46	INFC47	INFC48	INFC49	INFC50	INFC51	INFC52	INFC53	INFC54	INFC55	INFC56	INFC57	INFC58	INFC59	INFC60	INFC61	INFC62	INFC63	INFC64	INFC65	INFC66	INFC67	INFC68	INFC69	INFC70	INFC71	INFC72	INFC73	INFC74	INFC75	INFC76	INFC77	INFC78	INFC79	INFC80	INFC81	INFC82	INFC83	INFC84	INFC85	INFC86	INFC87	INFC88	INFC89	INFC90	INFC91	INFC92	INFC93	INFC94	INFC95	INFC96	INFC97	INFC98	INFC99	INFC100	INFC101	INFC102	INFC103	INFC104	INFC105	INFC106	INFC107	INFC108	INFC109	INFC110	INFC111	INFC112	INFC113	INFC114	INFC115	INFC116	INFC117	INFC118	INFC119	INFC120	INFC121	INFC122	INFC123	INFC124	INFC125	INFC126	INFC127	INFC128	INFC129	INFC130	INFC131	INFC132	INFC133	INFC134	INFC135	INFC136	INFC137	INFC138	INFC139	INFC140	INFC141	INFC142	INFC143	INFC144	INFC145	INFC146	INFC147	INFC148	INFC149	INFC150	INFC151	INFC152	INFC153	INFC154	INFC155	INFC156	INFC157	INFC158	INFC159	INFC160	INFC161	INFC162	INFC163	INFC164	INFC165	INFC166	INFC167	INFC168	INFC169	INFC170	INFC171	INFC172	INFC173	INFC174	INFC175	INFC176	INFC177	INFC178	INFC179	INFC180	INFC181	INFC182	INFC183	INFC184	INFC185	INFC186	INFC187	INFC188	INFC189	INFC190	INFC191	INFC192	INFC193	INFC194	INFC195	INFC196	INFC197	INFC198	INFC199	INFC200	INFC201	INFC202	INFC203	INFC204	INFC205	INFC206	INFC207	INFC208	INFC209	INFC210	INFC211	INFC212	INFC213	INFC214	INFC215	INFC216	INFC217	INFC218	INFC219	INFC220	INFC221	INFC222	INFC223	INFC224	INFC225	INFC226	INFC227	INFC228	INFC229	INFC230	INFC231	INFC232	INFC233	INFC234	INFC235	INFC236	INFC237	INFC238	INFC239	INFC240	INFC241	INFC242	INFC243	INFC244	INFC245	INFC246	INFC247	INFC248	INFC249	INFC250	INFC251	INFC252	INFC253	INFC254	INFC255	INFC256	INFC257	INFC258	INFC259	INFC260	INFC261	INFC262	INFC263	INFC264	INFC265	INFC266	INFC267	INFC268	INFC269	INFC270	INFC271	INFC272	INFC273	INFC274	INFC275	INFC276	INFC277	INFC278	INFC279	INFC280	INFC281	INFC282	INFC283	INFC284	INFC285	INFC286	INFC287	INFC288	INFC289	INFC290	INFC291	INFC292	INFC293	INFC294	INFC295	INFC296	INFC297	INFC298	INFC299	INFC300	INFC301	INFC302	INFC303	INFC304	INFC305	INFC306	INFC307	INFC308	INFC309	INFC310	INFC311	INFC312	INFC313	INFC314	INFC315	INFC316	INFC317	INFC318	INFC319	INFC320	INFC321	INFC322	INFC323	INFC324	INFC325	INFC326	INFC327	INFC328	INFC329	INFC330	INFC331	INFC332	INFC333	INFC334	INFC335	INFC336	INFC337	INFC338	INFC339	INFC340	INFC341	INFC342	INFC343	INFC344	INFC345	INFC346	INFC347	INFC348	INFC349	INFC350	INFC351	INFC352	INFC353	INFC354	INFC355	INFC356	INFC357	INFC358	INFC359	INFC360	INFC361	INFC362	INFC363	INFC364	INFC365	INFC366	INFC367	INFC368	INFC369	INFC370	INFC371	INFC372	INFC373	INFC374	INFC375	INFC376	INFC377	INFC378	INFC379	INFC380	INFC38
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	--------



LDA	FDTS1R	LOAD A WITH FILE DESCRIPTION TABLE INACTIVE.
SKA	BITO	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
ARM	\$-2	
BRU	R/I OPS	
P7E	1	READ
OPEDT	0,040,FDTS1W	SAVE LISTING
LDA	FDTS1W	FROM CORE ONTO THE RAD
SKA	BITO	LOAD A WITH FILE DESCRIPTION TABLE INACTIVE.
ARM	\$-2	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
BRU	R/I OPS	
P7E	1	WRITE
OPEDT	0,031,FDTS1P	END OF FILE MARK ON RAD
LDA	FDTS1P	AT END OF TRANSMISSION.
SKA	BITO	LOAD A WITH FILE DESCRIPTION TABLE INACTIVE.
ARM	\$-2	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
BRU	R/I OPS	
P7E	1	REWIND TO
OPEDT	0,032,FDTP1P	THE BEGINNING OF
LDA	FDTP1P	USER PROGRAM ON THE RAD.
SKA	BITO	LOAD A WITH FILE DESCRIPTION TABLE INACTIVE.
ARM	\$-2	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
BRU	R/I OPS	
P7E	1	WRITE
OPEDT	0,00,FDTP1C	PROGRAM LISTING
LDA	FDTP1C	INTC CORE FROM THE RAD.
SKA	BITO	LOAD A WITH FILE DESCRIPTION TABLE INACTIVE.
ARM	\$-2	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
BRU	\$+31	
BRU	R/I OPS	
P7E	1	REWIND TO
OPEDT	0,032,FDTS2R	THE BEGINNING OF
LDA	FDTS2R	SAVED DATA ON THE RAD
SKA	BITO	LOAD A WITH FILE DESCRIPTION TABLE INACTIVE.
ARM	\$-2	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
BRU	R/I OPS	
P7E	1	READ
OPEDT	0,040,FDTS2W	SAVE LISTING
LDA	FDTS2W	FROM CORE ONTO THE RAD
SKA	BITO	LOAD A WITH FILE DESCRIPTION TABLE INACTIVE.
ARM	\$-2	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
BRU	R/I OPS	
P7E	1	WRITE
OPEDT	0,031,FDTS2P	END OF FILE MARK ON RAD
LDA	FDTS2P	AT END OF TRANSMISSION.
SKA	BITO	LOAD A WITH FILE DESCRIPTION TABLE INACTIVE.
ARM	\$-2	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
BRU	R/I OPS	
P7E	1	REWIND TO
OPEDT		THE BEGINNING OF

OPENT	0,032,F0TP2R	USER PROGRAM ON THE RAD.
LDA	F0TP2R	LOAD A WITH FILE DESCRIPTION TABLE INACTIVE.
SKA	R1T0	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
ARM	\$-2	
P7E	R\ICPS	WRITE
OPENT	1	PROGRAM LISTING
LDA	0,00,F0TP2C	INTC CORR FROM THE RAD.
SKA	F0TP2C	LOAD A WITH FILE DESCRIPTION TABLE INACTIVE.
ARM	R1T0	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
DTR	\$-2	
SKN	RUGGY,1	SKIP IF DRUMS IS CALLING DRUM
RPU	\$+8	
LDA	ENDING,1	CALCULATE PAGE ADDRESS FOR I/O OPS.
ADD	ONE	
MUL	FOUP40	
ADD	CRIGIN	
LDR	X2RIT	MERGE A INTO B
CCPY	(5,4,4)	
STB	BUILD,1	
LDX	=0177110,2	
SKN	RFAD,1	SKIP IF DRUMRD CALLED
BRU	\$+2	
ARM	\$+5	
LDA	*DINFO,1	
STA	*BLILD,1	
ARX	\$-2,2	
ARM	\$+4	
LDA	*BLILD,1	
STA	*DINFO,1	
ARX	\$-2,2	
EXU	\$+1,1	
BRU	\$+35	
SKN	RFAD,1	SKIP IF DRUMRD WAS CALLED
ARM	\$+2	
ARM	\$+10	
ARM	R\ICPS	REWIND TO
P7E	1	THE BEGINNING OF
OPENT	0,032,F0TP2R	USER PROGRAM ON THE RAD.
LDA	F0TP2R	LOAD A WITH FILE DESCRIPTION TABLE INACTIVE.
SKA	R1T0	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
ARM	\$-2	
P7E	R\ICPS	READ
OPENT	1	PROGRAM LISTING
LDA	0,040,F0TP2C	FROM CORE INTO THE RAC LISTING OF U.
SKA	F0TP2C	LOAD A WITH FILE DESCRIPTION TABLE INACTIVE.
ARM	R1T0	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
P7E	\$-2	



ARM	P/ICPS	WRITE	END OF FILE MARK ON RAD
DZE	1	END OF FILE TRANSMISSION.	
OPENT	0,031,FNT020	AT END OF TRANSMISSION.	
LDA	FNT020	LOAD A WITH FILE DESCRIPTION	ACTIVITY WRD
SKA	RTTO	SKIP IF FILE DESCRIPTION TABLE INACTIVE.	
ARM	4-2		
ARM	P/ICPS	REWIND TO	
DZE	1	THE BEGINNING OF	
OPENT	0,032,FNTS20	SAVED DATA ON THE RAD	
LDA	FNTS20	LOAD A WITH FILE DESCRIPTION	ACTIVITY WRD
SKA	RTTO	SKIP IF FILE DESCRIPTION TABLE INACTIVE.	
ARM	4-2		
ARM	P/ICPS	WRITE	
DZE	1	SAVE LISTING	
OPENT	0,00,FNTS2W	FROM RAD IN CORE.	
LDA	FNTS2W	LOAD A WITH FILE DESCRIPTION	ACTIVITY WRD
SKA	RTTO	SKIP IF FILE DESCRIPTION TABLE INACTIVE.	
ARM	4-2		
ARM	4-34		
SKN	FEAD,1	SKIP IF CRIMRD WAS CALLED	
ARM	4-2		
ARM	4-10		
ARM	P/ICPS	REWIND TO	
DZE	1	THE BEGINNING OF	
OPENT	0,032,FNT010	USER PROGRAM ON THE RAD.	ACTIVITY WRD
LDA	FNT010	LOAD A WITH FILE DESCRIPTION	TABLE INACTIVE.
SKA	RTTO	SKIP IF FILE DESCRIPTION	
ARM	4-2		
ARM	P/ICPS	READ	
DZE	1	PROGRAM LISTING	
OPENT	0,040,FNT01C	FROM CORE INTO THE RAD	LISTING OF U.
LDA	FNT01C	LOAD A WITH FILE DESCRIPTION	ACTIVITY WRD
SKA	RTTO	SKIP IF FILE DESCRIPTION	TABLE INACTIVE.
ARM	4-2		
ARM	P/ICPS	WRITE	
DZE	1	END OF FILE MARK ON RAD	
OPENT	0,031,FNT010	AT END OF TRANSMISSION.	
LDA	FNT010	LOAD A WITH FILE DESCRIPTION	ACTIVITY WRD
SKA	RTTO	SKIP IF FILE DESCRIPTION	TABLE INACTIVE.
ARM	4-2		
ARM	P/ICPS	REWIND TO	
DZE	1	THE BEGINNING OF	
OPENT	0,032,FNTS1R	SAVED DATA ON THE RAD	
LDA	FNTS1R	LOAD A WITH FILE DESCRIPTION	ACTIVITY WRD
SKA	RTTO	SKIP IF FILE DESCRIPTION	TABLE INACTIVE.
ARM	4-2		
ARM	P/ICPS	WRITE	
DZE	1	SAVE LISTING	





PCH	C34001		
STA	R X3,1		
STR	R,1	NUMBER OF ARGUMENTS	
LDA	ARGNUM	NUMBER OF UNREAL ARGUMENTS	
SUR	ONE	SAVE	
STA	TEMPARG,1		
SKU	ZFRO	SKIP IF MORE ARGUMENTS TO BE READ	
SRU	\$+20		
COPY	(0,5)	CLEAR A	
SUR	TEMPARG,1	SET UP X2 REGISTER ARG.	
RCH	034001	STORE AND READ POINTERS	
MRC	=0100000		
COPY	(5,2)		
STA	TEMPOX2,1	SAVE ARG X2 CONTROLLER	
LDX	=0100002,3		
LDA	STEPY	SET UP FOR READING THE VARIABLE ADDRESSES	
MRC	=020000000		
STA	STEPY		
RRX	\$+1,3		
LDA	*STEPY	BRING IN THE ARGUMENTS	
STA	*ARGANS,1		
RRX	\$-4,2		
LDA	TEMPOX2,1	ARG. POINTER	
COPY	(5,2)		
LDA	*IARGADS,1	BRING IN THE CONTENTS ON THE ADDRESS	
STA	*ARGS,1	ARGS AND SAVE THEM	
RRX	\$-2,2		
LDA	STEPY		
RCH	034001	SAVE ADDRESS PORTION	
STA	*FDTPC,1		
MFO	ARGNUM		
ADD	LOC,1	SET UP STARTING ADDRESS FOR STEPPING	
STA	=07777	EXECUTABLE INSTRUCTION AND SAVE.	
LDA	*FDTPN,1	MAX PROGRAM SIZE.	
COPY	(0,5)		
STA	FLAGWRD,1	CLEAR FLAG STORAGE AREA.	
STA	FLAG,1		
LDA	ONE		
STA	FIRSPAS,1	RESET PROGRAM INITIALIZATION FLAGS.	
LDA	MINUS1		
STA	ISHT,1		
RRM	KEYO	INITIALIZE AND START UP SCOPE.	
EXU	\$+1,1	SAVE UNEXECUTED PROGRAM FOR LATER STEP BY	
RRU	\$+20	STEP-BY-STEP EXECUTION AT SCOPE CONSOLE.	
RRM	R/10PS		
DZE	1		
OFFPT	0,032,FDTD2R		

LDA	FNTP2R	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
SKA	RTTO	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
APM	\$-2	
PZE	RNIOps	
NPENT	1	
LDA	0,040,FNTP2C	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
SKA	FNTP2C	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
APU	RTTO	
BRM	\$-2	
PZE	RNIOps	
NPENT	1	
LDA	0,031,FNTP2R	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
SKA	FNTP2R	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
APU	RTTO	
ARM	\$-2	
ARM	\$+1q	
PZE	RNIOps	
NPENT	1	
LDA	0,032,FNTP1P	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
SKA	FNTP1P	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
APU	RTTO	
ARM	\$-2	
ARM	RNIOps	
PZE	1	
NPENT	0,040,FNTP1C	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
LDA	FNTP1C	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
SKA	RTTO	
ARM	\$-2	
ARM	RNIOps	
PZE	1	
NPENT	0,031,FNTP1P	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
LDA	FNTP1P	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
SKA	RTTO	
ARM	\$-2	
ARM	STEPRY	
PZE	INTERRUPT SERVICER FOR EACH STEP	
NPENT		
LDA	MINUS1	SFT HIGHER INTERRUPT PRIORITY FLAG.
STA	\$SECOND,1	
APU	\$+8	REDUCE THE INSTRUCTION POINTER BY ONE.
BACKUP1	LCC,1	
LDA	ONE	
STA	LCC,1	
APU	\$+2	
MPN	LCC,1	
LDA	ONE	INCREMENT THE INSTRUCTION POINTER
STA	FIRSTPAS,1	AND DISPLAY THE NEXT INSTRUCTION.
SKN	1SHOT,1	SKIP IF THIS PROGRAM WAS ACTIVATED



BRU	DEPART	FROM CALLERS PROGRAM, 0.W. EXIT.
SKN	PENCLR,1	SKIP IF WERE ANY CHANGES TO THE DISPLAY
BRU	\$+2	OF THE NEXT INSTRUCTION OR REGISTERS.
EXU	BRINGIN	
BRU	\$+1,1	SAVE PROGRAM AREA AND WRITE UNEXECUTED
BRM	\$+32	PROGRAM IN FOR 1-STEP EXECUTION.
BRM	RNIOPS	
P7E	1	
OPFNT	0,032,FDT2R	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
LDA	FDT2R	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
SKA	RIT0	
BRU	\$-2	
BRM	RNIOPS	
P7E	1	
OPFNT	0,040,FDT2W	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
LDA	FDT2W	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
SKA	RIT0	
BRU	\$-2	
BRM	RNIOPS	
P7E	1	
OPFNT	0,031,FDT2R	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
LDA	FDT2R	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
SKA	RIT0	
BRU	\$-2	
BRM	RNIOPS	
P7E	1	
OPFNT	0,032,FDT2R	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
LDA	FDT2R	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
SKA	RIT0	
BRU	\$-2	
BRM	RNIOPS	
P7E	1	
OPFNT	0,00,FDT2C	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
LDA	FDT2C	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
SKA	RIT0	
BRU	\$-2	
BRU	\$+31	
BRM	RNIOPS	
P7E	1	
OPFNT	0,032,FDT2R	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
LDA	FDT2R	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
SKA	RIT0	
BRU	\$-2	
BRM	RNIOPS	
P7E	1	
OPFNT	0,040,FDT2W	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
LDA	FDT2W	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
SKA	RIT0	

RRU	\$-2	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
ARM	R\NOPS	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
07E	1	
OPENT	0,031,FNTSIR	
LDA	FNTSIR	
SKA	R\TO	
ARM	\$-2	
ARM	R\NOPS	
07E	1	
OPENT	0,032,FNTPIR	
LDA	FNTPIR	
SKA	R\TO	
ARM	\$-2	
ARM	R\NOPS	
07E	1	
OPENT	0,00,FDTPIR	
LDA	FDTPIR	
SKA	R\TO	
ARM	\$-2	
DIR	1	
SKN	FIRSPAS,1	
ARM	PRESENT	
SKN	NFWINST,1	
ARM	\$+3	
LDA	#LCC,1	
STA	INSTR,1	
LDR	=07700000	
SKM	HLT	
ARM	\$+3	
MPO	LOC,1	
ARM	PRESENT	
SKM	EXU	
ARM	\$+12	
ARM	ADRCALC	
SKN	EXUFLG,1	
ARM	\$+2	
ARM	\$+5	
LDA	MINUS1	
STA	EXUFLG,1	
LDA	INSTR,1	
STA	TEMPLOC,1	
LDA	FECDADR,1	
STA	INSTR,1	
ARM	RESTR+2	
SKM	ARM	
ARM	\$+4	
ARM	ADRCALC	
ARM	TRYEXU	

RESTRT		LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
		SKIP IF FILE DESCRIPTION TABLE INACTIVE.
		LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
		SKIP IF FILE DESCRIPTION TABLE INACTIVE.
		SKIP IF THIS IS NOT THE FIRST PASS THRU.
		SKIP IF INSTRUCTION REGISTER (DISPLAY)
		WAS CHANGED BY LIGHT PEN STRIKE.
		CHECK FOR AN INSTRUCTION TYPE THAT
		ALTERS THE P REGISTER.
		SKIP IF THE INSTRUCTION IS A HALT.
		PROCESS HALT.
		SKIP IF EXECUTE (EXU)
		CALCULATE THE EFFECTIVE ADDRESS.
		SKIP IF MULTIPLE-LEVEL EXUS.
		SET MULTIPLE-EXU FLAG.
		SAVE PRESENT ADDRESS FOR RETURN AFTER EX
		BRING IN THE EFFECTIVE ADDRESS OF
		THE EXU AND PROCESS IT.
		SKIP IF INSTRUCTION IS A BRU.
		PRECCESS THE BRU. CALCULATE EFFECTIVW
		ADDRESS AND TRY TO EXECUTE IT.

BRU	EXUINST	SKIP IF INST. IS A BRX OR BRC.
SKM	BRXC	
ARU	\$+15	
SKA	INDEX	SKIP IF IS A BRC
ARU	\$+10	
SKA	RITO	SKIP IF NOT INDIRECT ADDRESSED.
ARU	\$+4	
RCH	034001	CLEAR BITS 0-8 INCLUSIVE:
STA	EFCADR,1	RESULT IS EFFECTIVE ADDRESS STORE IT.
ARU	GETSET	
RCH	034001	CLEAR BITS 0-8 INCLUSIVE
STA	TLCC,1	
LDA	*TLCC,1	
ARU	\$-8	
LDA	MINUS1	
STA	RPXFLG,1	
ARU	\$-11	SKIP IF INSTR. IS A BRM.
SKM	ARM	
ARU	\$+5	
ARM	ADRCALC	
MPO	EFCADR,1	SKIP IF INSTR. IS A BRR.
ARM	TPYEXU	PROCESS IT)
BRU	EXUINST	
SKM	RRR	CLEAR BITS 0-8 INCLUSIVE.
ARM	\$+8	
LDA	ADRCALC	
ADD	*EFCADR,1	SKIP IF INSTR. IS A BMA.
RCH	ONE	
STA	034001	CLEAR ALL FLAGS
ARM	EFCADR,1	SKIP IF AN EXU INSTRUCTION.
SKM	TPYEXU	RESTORE EXU INSTRUCTION ADDRESS TO
BRU	EXUINST	INSTRUCTION TO BE EXECUTED POINTER.
ARM	BMA	RESET EXU FLAG.
ARU	\$+2	
BRU	ARMDC	PUT RETURN ADDRESS IN FLAGWRD
EXUINST	07700	BRING IN STEP-BY-STEP PROG. FLAGS.
NCP	EXUFLG,1	SET UP REGISTERS WITH STEP-BY-STEP
SKN	\$+4	PROGRAMS VALUES FOR EXECUTIONZ.
ARU	TEMPLOC,1	
LDA	INSTR,1	
STA	EXUFLG,1	
MPT	= \$+3	
LDA	ADDR	
LDR	FLAGWRD,1	
STS	FLAGWRD,1	
ARR	B,1	
LDB	X3,1	
LDA		

COPY	(5,3)	PROCEED TO RESPECTIVE EXECUTION PROGRAM
LDA	X2,1	AND EXECUTE THE NEXT INSTRUCTION.
COPY	(5,2)	
EXU	\$+1,1	
BRU	\$+9	
COPY	(1,5)	
STA	DISPNC2	
LDA	X1,1	
COPY	(5,1)	
LDA	A+1	
EXU	INSTR+1	
BRU	NOSKIP2	EXECUTION RESULTED IN NO SKIP, PROCESS
BRU	SKIP2	EXECUTION RESULTED IN A SKIP, PROCESS IT.
LDA	MINUS1	
SKU	PCNG,1	SKIP IF NO P REGISTER CHANGES.
STA	DISFLG,1	
COPY	(1,5)	
STA	DISPN01	DISPLAY NUMBER 1 EXECUTION ROUTINE.
LDA	X1,1	
COPY	(5,1)	
LDA	A	
EXU	INSTR	
BRU	NOSKIP1	RESTORE X1
XMV	DISPN01,1	ADD 2 TO THE LOCATION POINTER TO REFLECT
MPT	LOC,1	THE SKIP ON EXECUTION.
BRU	\$+3	RESTORE X1
NOSKIP1	DISPN01,1	ADD 1 TO THE INSTRUCTION POINTER TO REFLECT
MPO	LOC,1	NO SKIP ON EXECUTION.
STA	A	
LDA	DISPN01	
BRU	\$+14	RESTORE X1
NOSKIP2	DISPN02,1	
MPO	LOC,1	
BRU	\$+3	
XMV	DISPNC2,1	
MPT	LOC,1	
STA	A	
LDA	DISPN02	
SKU	PCNG,1	
BRU	\$+5	
MPT	DISFLG,1	
SKU	RRXFLG,1	
BRU	\$+2	
MPT	RRXFLG,1	
STA	X1,1	
STB	R,1	SAVE PROGRAM REGISTERS.
COPY	(2,5)	
STA	X2,1	

COPY	(3,5)	BRING IN NEW
STA	X3,1	CONTENTS OF
LDA	TEMPX2,1	OF THE SPECIFIED
COPY	(5,2)	TRACE VARIABLES
LDA	*TARGADS,1	
STA	*ARGS,1	
RRX	\$-2,2	PUT FLAG REGISTER IN BITS 0-8 INCLUSIVE
RPM	\$+1	
PZE	0	SAVE THE CONTENTS OF THE FLAG REGISTER IN
LDA	\$-1	IN FLAGWCRD.
STA	FLAGWPD,1	CLEAR THE FLAG REGISTER
FSTR	07700	
NOP		SKIP IF EFFECTIVE ADDRESS WAS CHANGED.
SKN	PCNG,1	RESET EFFECTIVE ADDRESS ALTERED FLAG.
RRU	PCNG,1	
MPT	AFTAXU	SAVE NEXT INSTRUCTION ADDRESS IN LOC
LDA	034001	FOR NEXT EXECUTION.
RCH	LOC,1	PUT THE UNALTERED CONTENTS OF THE
STA	CONFEC,1	EFFECTIVE ADDRESS BACK.
LDA	*EFCADR,1	SKIP IF SEVERAL ADDRESSES WERE AFFECTD
STA	TRAC,1	
SKN	\$+10	RESET FLAG FOR ALTERED ADDRESSES RESTORED
RRU	TRAC,1	SET UP X3 REGISTER FOR
MPT	SVX3,1	CONTROLLING NUMBER OF
LDA	034007	ALTERED ADDRESS TO BE RESTORED.
RCH	(5,3)	(LIMIT IS FIVE)
COPY	MTNUS1	DECREMENT EFCADR BY 1.
LDA	EFCADR,1	RESTORE ORIGINAL CONTENTS TO THE
ADM	*SVFFCA,1	ALTERED ADDRESS.
LDA	*EFCADR,1	
STA	\$-4,3	CHECK FOR POSSIBLE ERRORS DUE TO
RRX	CADR,1	FIVE OR MORE LEVELS OF INDIRECT ADDRESS
LDA	EFCADR,1	
SKE	DEFERR	
RRU	PRESENT	
RRU	0	
ADRCALC	PZE	
LDX	=0100000,3	
LDX	=0177766,2	
CCPY	(0,5)	
STA	*INTAD,1	RESET TO TEMP STORAGE LOCATIONS.
RRX	\$+1,3	
RRX	\$-2,2	RESET POINTERS AND ITERATION CONTROLS.
LDA	=0100000,3	
LDA	FIVE	
STA	RUNCU,1	
COPY	(0,5)	
STA	PART,1	



LDA  
SKA  
RPU  
BRU  
SKR  
BRU  
ARM  
LRSA  
ETR  
SUB  
COPY  
LDA  
COPY  
SKN  
BRX  
RCH  
ADD  
RCH  
LOB  
SKN  
STS  
STS  
SKA  
BRU  
BRU  
SKN  
BRX  
LDA  
STS  
RCH  
STA  
STA  
LDA  
SKE  
BRU  
LDA  
STA  
BRU  
PTE  
SKN  
BRU  
MPT  
BRU  
LDX  
LDX  
LDA  
SKE

TRYEXU

INSTR,1  
INDEX  
\$+2  
\$+10  
RUNCOU,1  
\$+2  
TRCUR  
21  
=03  
ONE  
(5,2)  
INSTR,1  
(5,4)  
RUNCOU,1  
\$+1,3  
034001  
\*X,2  
034003  
ADDR  
RUNCOU,1  
\*INTAD,1  
PART,1  
RITO  
\$+2  
\$+7  
RUNCOU,1  
\$+1,3  
\*PART,1  
\*INTAD,1  
\$-28  
034001  
EFCADR,1  
CADR,1  
RUNCOU,1  
FIVE  
\$+3  
MINUS1  
NOMOD,1  
NORCALC  
0  
NOMOD,1  
\$+3  
NOMOD,1  
GETSET  
=0177766,2  
=0100000,3  
EFCADR,1  
\*INTAD,1

CALCULATE THE EFFECTIVE ADDRESS.  
SKIP IF NO INDEXING USED.

SKIP IF MORE THAN 5 LEVELS OF INDIRECT ADDRESSING ARE USED.  
PUT INDEX VALUE IN X2 FOR POINTER CONTROL

SKIP IF MORE THAN 5 LEVELS OF INDIRECT INDEX AND INDIRECT BITS INTO A FROM B.  
MASK IN B  
SKIP IF MORE THAN FIVE LEVELS OF INDIRECT SAVE ADDRESS TO CHECK LATER FOR ALTERNATING SAVE FOR INDEX CHECKING.  
SKIP IF NO INDIRECT ADDRESSING USED.

SKIP IF 5 LEVELS ON INDIRECT ADDRESSING INCREMENT INDIRECT ADDRESS POINTER SAVED  
INDIRECT, SO CHECK AGAIN FOR INDEXING.  
CLEAR BITS 0-8 INCLUSIVE.  
STORE CALCULATED EFFECTIVE ADDRESS.  
SAVE CALCULATED EFFECTIVE ADDRESS FOR LATER CHECKING.  
RESET YTTTTTTTTTTTTTTTT  
SKIP IF NO INDEXING OR INDIRECTING.

SKIP IF NO INDEXING OR INDIRECTING.  
RESET NO INDEXING, INDIRECTING FLAG.

CHECK TO SEE IF THE EFFECTIVE ADDRESS IS USED TO CALCULATE THE EFFECTIVE ADDRESS

BRX	\$+2,3	BY THE PROGRAM, IF YES THEN PUT THE
BRX	\$+3,2	RETURN BRANCH IN THE NEXT UN-EFFECTIVE
BRX	\$+16	ADDRESS(NOPS IN AFFECTED ADDRESSES.
SKP	FOUR,1	
BRU	\$+2	
BRU	POSSIAL	
MPO	SVX3,1	
LDA	SVX3,1	
COPY	(5,3)	
LDA	*EFCADR,1	SAVE CONTENTS OF EFFECTIVE ADDRESS.
STA	*SVEFCA,1	
RCH	O14001	
MRG	NOP	
STA	*EFCADR,1	
MPO	EFCADR,1	
LDA	MINUS1	
STA	TRACK,1	
BRU	\$-21	SAVE CONTENTS OF THE FINAL EFFECTIVE
LDA	*EFCADR,1	ADDRESS AND THEN STORE A RETURN TO THE
STA	CONECF,1	STEP-RY-STEP PROGRAM
LDA	BRMTCP	IN THE EFFECTIVE ADDRESS.
STA	*EFCADR,1	
LDA	MINUS1	
STA	PCNG,1	SET CHANGED EFFECTIVE ADDRESS CONTENTS
LDA	=10	FLAG. RESET CONTENTS POINTER.
LDA	FOUR,1	
SKN	BRXELG,1	
BRR	TRYEXU	
BRU	EXUINST	
PZE	0	SKIP IF PROCASSING FROM DISLAY 1.
SKN	DISFLG	
BRU	\$+6	RESET DISPLAY 1 FLAG
MPT	DISFLG	RESTORE X1 FOR DISPLAY 1.
XXM	DISPN01,1	
STA	A1	
LDA	DISPNC1	RETURN TO NORMAL FINISH PROCESSOR
BRU	SAVNEW	PROCESS DISPLAY 2 AFTER EXECUTION.
XXM	DISPN02,1	
STA	A1	
LDA	DISPNC2	RETURN TO NORMAL FINISH
BRU	SAVNEW	
PRESENT	*LOC,1	
LDA	INSTR,1	RESTORE CORE TO PRE-INTERRUPTED STATE.
STA	\$+1,1	
EXU	\$+32	
BRU	R\IOPS	
BRM	1	
D7E		

OPEDT	0,032,FDTp2R	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
LDA	FDTp2R	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
SKA	RTTO	
BRU	\$-2	
BRM	R\IOPS	
DZE	1	
OPEDT	0,040,FDTp2C	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
LDA	FDTp2C	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
SKA	RTTO	
BRU	\$-2	
BRM	R\IOPS	
DZE	1	
OPEDT	0,031,FDTp2R	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
LDA	FDTp2R	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
SKA	RTTO	
BRU	\$-2	
BRM	P\IOPS	
DZE	1	
OPEDT	0,032,FDTs2R	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
LDA	FDTs2R	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
SKA	RTTO	
BRU	\$-2	
BRM	R\IOPS	
DZE	1	
OPEDT	0,00,FDTs2h	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
LDA	FDTs2h	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
SKA	RTTO	
BRU	\$-2	
BRU	\$+31	
BRM	R\IOPS	
DZE	1	
OPEDT	0,032,FDTp1R	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
LDA	FDTp1R	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
SKA	RTTO	
BRU	\$-2	
BRM	R\IOPS	
DZE	1	
OPEDT	0,040,FDTp1C	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
LDA	FDTp1C	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
SKA	RTTO	
BRU	\$-2	
BRM	R\IOPS	
DZE	1	
OPEDT	0,031,FDTp1R	LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
LDA	FDTp1R	SKIP IF FILE DESCRIPTION TABLE INACTIVE.
SKA	RTTO	
BRU	\$-2	
BRM	R\IOPS	

```

P7E      1 0,032,FDTSLR
LDA      FDTSLR
SKA      RITO
ARM      $-2
ARM      R>IOPS
P7E      1 0,00,FDTSLR
LDA      FDTSLR
SKA      RITO
ARM      $-2
LDX      =0177772,2
LDA      FLAGWRD,1
LLSA     2
SKA      1 RITO
ARM      $+2
ARM      $+6
LDB      MASK+6,2
LDS      FLAGNUM+6,2
STX      FLAG,1
ARM      $-7,2
LDB      $+4
LDS      MASK+6,2
LDS      ZFRC
$-5      =0100005,2
$-5      =0177751,3
SPACST   SPACST
SPACST   SPACST
FIVE     (2,5)
FIVE     (5,2)
$-4,3    SPACST
SPACST   SPACST
DISTSTEP,1
*CHAINIT,1
TEMPOX2,1
(5,2)    =0177777,3
$+1,3    =0177777,3
DARGC,1  $+1,3
*PARGC,1 DARGC,1
*PARGC,1 *PARGC,1
NSCON,1  $-5

```

SHN

```

LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
SKIP IF FILE DESCRIPTION TABLE INACTIVE.

LOAD A WITH FILE DESCRIPTION ACTIVITY WRD
SKIP IF FILE DESCRIPTION TABLE INACTIVE.

      TRASSFORM THE FLAG REGISTER TO
      A USEABLE FORM FOR DISPLAY
SKIP IF FLAG IS NOY LIT.

      SETUP TO CLEAR OLD
      DISPLAY AND
      GET READY FOR NEW.

      CLEAR OLD DISPLAY.

      START UP STEP-BY STEP DISPLAY.
      BRING IN ARG POINTERS

      SFT UP POINTERS AND
      BRING IN THE
      CONTENTS OF THE
      TRACE ARGUMENTS AND
      TRANSLATE THEN TO ASC II CODE FOR DISPLAY

```





ADD	SIST	
STA	DIRT,1	
LDA	*DIRT,1	
LLSA	06	
MKG	TEMPA,1	
STA	TEMPA,1	
COPY	(0,5)	
LLSD	03	
ADD	SIST	
STA	DIRT,1	
LDA	*DIRT,1	
MKG	TEMPA,1	
SKN	ARGTRA,1	
BRU	TRARET	
STA	*PERMAD,1	
SKR	SECWRD,1	
BRU	\$-34	
MPT	SECWRD,1	
RRX	\$+1,3	
LDA	SPACES	
STA	*PERMAD,1	
RRX	\$-46,2	
LDA	MINUS1	
STA	FIRSPAS,1	
STA	NEWINST,1	
BRU	DEPART	
RRINGIN	0	
MPT	PFNCOR,1	
LDX	=0177770,2	
LDB	=0100002,3	
RRX	*PERMAD,1	
COPY	\$+1,3	
LLSD	(0,5)	
ADD	06	
STA	FIRST	
LDA	*DIRT,1	
LLSA	21	
STA	TEMPA,1	
LLSD	06	
ADD	FIRST	
STA	*DIRT,1	
LDA	18	
MKG	TEMPA,1	
STA	06	
LLSD	FIRST	
ADD		

SKIP IF CONVERTING REGISTER ARGUMENTS  
 RESFT SECOND WORD FLAG  
 RESET THE FIRST PASS FLAG  
 CONVERT CHANGES MADE ON THE SCOPE  
 BACK FOR USE ON THE EXECUTION OF THE  
 STEP-BY-STEP PROGRAM.

STA	DIRT,1
LDA	#DIR,1
LLSA	15
MRG	TEMPA,1
STAY	(0,5)
COPY	(0,6)
LLSD	FIRST
ADD	FIRST,1
STA	#DIR,1
LDA	12
LLSA	TEMPA,1
MRG	TEMPA,1
STA	*PERMAD,1
LDB	(0,5)
COPY	(0,6)
LLSD	FIRST
ADD	FIRST,1
STA	#DIR,1
LDA	9
LLSA	TEMPA,1
MRG	TEMPA,1
STAY	(0,5)
COPY	(0,6)
LLSD	FIRST
ADD	FIRST,1
STA	#DIR,1
LDA	6
LLSA	TEMPA,1
MRG	TEMPA,1
STAY	(0,5)
COPY	(0,6)
LLSD	FIRST
ADD	FIRST,1
STA	#DIR,1
LDA	3
LLSA	TEMPA,1
MRG	TEMPA,1
STAY	(0,5)
COPY	(0,6)
LLSD	FIRST
ADD	FIRST,1
STA	#DIR,1
LDA	1
LLSA	TEMPA,1
MRG	*PERMAD,1
STAY	T+8,2
COPY	NXT,2
LLSD	CNE
ADD	NEWINST,1
STA	

TR0LB	RRR	RRINGIA	
PZE	STA	O	
COPY	COPY	TA,1	
COPY	COPY	(3,5)	
STA	STA	TX3,1	
LDX	STA	TX2,1	
LDX	LDA	=0177762,2	
LDA	LDA	=0100000,3	
STA	STA	MSG+14,2	
RRX	RRX	*SBI,1	
RRX	RRX	+1,3	
LDA	RRX	-3,2	
COPY	LDA	TX3,1	
LDA	COPY	(5,3)	
COPY	LDA	TX2,1	
LDA	COPY	(2,5)	
RRR	LDA	TA,1	
POSSIAL	RRR	TR0UR	
LDX	LDA	=0177761,2	
LDA	LDA	=0100000,3	
STA	STA	MSG+15,2	
RRX	RRX	*SBI,1	
RRX	RRX	+1,3	
RRU	RRU	-3,2	
DEFERR	RRU	IGNORE	
LDX	LDA	=0177756,2	
LDA	LDA	=0100000,3	
STA	STA	MSG+18,2	
RRX	RRX	*SBI,1	
RRU	RRU	+1,3	
RRU	RRU	-3,2	
PRESENT	RRU	PRESENT	
DATA	DATA	1,1	
DATA	DATA	0,1	
DATA	DATA	00300000	
DATA	DATA	04100000	
DATA	DATA	04300000	
DATA	DATA	05700000	
DATA	DATA	02100000	
DATA	DATA	0	
DATA	DATA	01C00000	
DATA	DATA	0,0	
DATA	DATA	1,1	
DATA	DATA	1,1	
DATA	DATA	1,1	
DATA	DATA	0,0	
DATA	DATA	03C0000000	

SAVE PREGRAM IN TEMPORARY LOCATIONS.

SET UP POINTERS AND TRANSMITT THE  
ERROR MESSAGE TO THE DISPLAY.  
-MORE THAN 5 LEVELS OF INDIRECT/INDEXING  
POSSIBLE ERROR.

SET UP POINTERS AND TRANSMITT ERROR  
MESSAGE.  
- NOT ABLE TO PROCESS EXCESSINDIRECT/  
INDEXING, STEP IGNORED-

SET UP THE POINTERS AND TRANSMITT THE  
ERROR MESSAGE.  
-THE PREVIOUS STEP HAS = 5 LEVELS OF  
INDIRECT/INDEXING, ERROR GENERATED.-

RITC	DATA	04C0000000
X	P7E	RX1,1
	PZE	PX2,1
	PZE	PX3,1
	DRES	0,0
PART	DRES	2
FFCADR	P7E	INTAD1,3
INTAR	P7E	INTAD2,3
	DATA	0,0,0,0,0,0,0,0,0,0
INTAD1	DATA	0,0,0,0,0,0,0,0,0,0
INTAD2	DATA	0,0,0,0,0,0,0,0,0,0
RUNCCU	DATA	5,5
NOMCD	DATA	1,1
SVEFCA	P7E	FFAD1+1,3
	P7E	FFAD2+1,3
EFAD1	DATA	0,0,0,0,0,0,0,0,0,0
EFAD2	DATA	0,0,0,0,0,0,0,0,0,0
CVX3	DATA	2,-2
CADR	RES	2
FOUR	DATA	10,10
TRACK	DATA	1,1
PCNGTQ	DATA	1,1
BRMCT	DATA	AFTAXU
OPFCT	DATA	1,8,15
EDTS1W	DATA	0,0,0,0,0,0,0,0,0,0
EDTS2W	DATA	0,0,0,0,0,0,0,0,0,0
EDTP1C	DATA	0,0,0,0,0,0,0,0,0,0
EDTP2C	DATA	0,0,0,0,0,0,0,0,0,0
EDTP1R	DATA	0,0,0,0,0,0,0,0,0,0
EDTP2R	DATA	0,0,0,0,0,0,0,0,0,0
EDTS1R	DATA	0,0,0,0,0,0,0,0,0,0
EDTS2R	DATA	0,0,0,0,0,0,0,0,0,0
ADDR	DATA	0,0,0,0,0,0,0,0,0,0
ADTPC	DATA	0,0,0,0,0,0,0,0,0,0
EDTPN	DATA	EDTP1C+1,EDTP2C+1
QUICKA	RES	EDTP1C+2,EDTP2C+2
QUICKA	RES	1
TX2	RES	1
TX3	RES	2
PX1	RES	2
PX2	RES	2
PX3	RES	2
A	RES	2
R	RES	2
X1	RES	2
X2	RES	2
X3	RES	2
LOC	RES	2

ARGNUM	RES	SS	1	
TEMPX2	RES	SS	2	
TEMPARG	PZE			
ARGC	PZE			ARGD1+6,2
				ARGD2+6,2
ARGD1	RES		6	
ARGD2	RES		6	
ARGADS	PZE			ARGADD1+6,2
	PZE			ARGADD2+6,2
ARGADD1	RES	SS	6	
ARGADD2	RES	SS	6	
IARGADS	PZE			*ARGADD1+6,2
	PZE			*ARGADD2+6,2
DARGC	DATA			DIARGS-1,DIARGS-1
DIARGS	DATA			01220761,040754040,01220762,040754040,01220763,040754040,
				01220764,040754040,01220765,040754040,01220766,040754040,
SHARGC	PZE			*D1SHARG+6,2
	PZE			*D2SHARG+6,2
D1SHARG	PZE			SBDAD12,3
	PZE			SBDAD13,3
	PZE			SBDAD14,3
	PZE			SBDAD15,3
	PZE			SBDAD16,3
	PZE			SBDAD17,3
D2SHARG	PZE			SBBAD12,3
	PZE			SBBAD13,3
	PZE			SBBAD14,3
	PZE			SBBAD15,3
	PZE			SBBAD16,3
	PZE			SBBAD17,3
NSCCN	DATA			1,1
ARGTRA	DATA			-1,1
PERMS	DATA			040204075,040114075,04140107,040024075,040014075,
				030614075,030624075,030634075
PERMDAT	PZE			LNCIF,1
	PZE			IFLAG,1
	PZE			R,1
	PZE			A,1
	PZE			X1,1
	PZE			X2,1
	PZE			X3,1
PERMAN	PZE			*PERMADD+9,2
	PZE			*PERRMADD+9,2
PERMAN	PZE			SBBAD1,3
	PZE			SBBAD2,3
	PZE			SBBAD3,3
	PZE			SBBAD4,3





```

-1
MINUS1 DATA
ONE DATA
FIVE DATA
DISPNO1 RES
DISPNO2 RES
CCONNEFC RES
END
*****
** FLASH AND FLASH2 **
*****
FLASH RESTARTS THE DISPLAY WHEN THE END OF THE ACTIVE LIST IS
REACHED, ALLOWING REFRESH AT SLIGHTLY LESS THAN THE MAX. RATE
*****
$FLASHI PZF          0      07601
XMX SVXD1,1
BRU $+4
$FLASH2 PZE          0      SVXD2,1
XMX SVXD2,1
FIRS 07700
EXU EOMPOT,1
PCT *PCTWRD,1
FSTR OLOI
BRU $+3
XMX SVXD2,1
BRC *FLASH2
XMX SVXD1,1
BRC *FLASH1
SVXD1 DATA          0
SVXD2 DATA          1
*****
STAR1 AND STAR2 *****
*****
INFO1 RES           0670
INFO1P RES         010
INFO2 RES           0670
INFO2P RES         010
*****
DISPLAY DATA LISTING *****
$STAR1 CWM        0,2,$+1

```

\$SRDAD1	DATA	08,0,1782,0
\$SRDAD2	CCRM	1,21,\$+1
	MMEM	0,2,0,8
	RES	3
\$SRDAD3	DATA	08,0,1739,0
\$SRDAD4	CCRM	0,21,\$+1
	MMEM	0,2,0,8
	RES	3
\$SRDAD5	DATA	08,0,1694,0
\$SRDAD6	CCRM	0,21,\$+1
	MMEM	0,2,0,8
	RES	3
\$SRDAD7	DATA	08,0,1650,0
	CCRM	0,21,\$+1
	MMEM	0,2,0,8
	RES	12
\$SRDAD8	DATA	08,0,1606,0
	CCRM	0,21,\$+1
	MMEM	0,2,0,8
	RES	12
\$SRDAD9	DATA	08,0,1562,0
	CCRM	0,21,\$+1
	MMEM	0,2,0,8
	RES	19
\$SRDAD10	DATA	08,0,1519,0
	CCRM	0,21,\$+1
	MMEM	0,2,0,8
	RES	19

\$SRDAD10	RES	0,2,0,8
	19	
	0,2,\$+1	
	0	
	88,0,1474,0	
	0,21,\$+1	
	0,2,0,8	
\$SRDAD11	RES	19
	0,2,\$+1	
	0	
	88,0,1430,0	
	0,21,\$+1	
	0,2,0,8	
\$SRDAD12	RES	26
	12	
	0,2,\$+1	
	0	
	88,0,1386,0	
	0,21,\$+1	
	0,2,0,8	
\$SRDAD14	RES	26
	12	
	0,2,\$+1	
	0	
	88,0,1342,0	
	0,21,\$+1	
	0,2,0,8	
\$SRDAD16	RES	26
	12	
	0,2,\$+1	
	0	
	88,0,1298,0	
	0,21,\$+1	
	0,2,0,8	
\$SRDAD17	RES	20
	0,2,\$+1	
	0	
	88,0,1254,0	
	0,21,\$+1	
	0,2,0,8	
	20	
	0,2,\$+1	
	0	
	88,0,1254,0	
	0,21,\$+1	
	0,2,0,8	
	20	
	0,2,\$+1	

DATA	08,0,1210,0
CCRFM	0,21,\$+1
CMFEM	0,2,0,8
RES	20
CMFEM	0,2,\$+1
DATA	08,0,1166,0
CCRFM	0,21,\$+1
CMFEM	0,2,0,8
RES	20
CMFEM	0,2,\$+1
DATA	08,0,1122,0
CCRFM	0,21,\$+1
CMFEM	0,2,0,8
RES	20
CMFEM	0,2,\$+1
DATA	08,0,1078,0
CCRFM	0,21,\$+1
CMFEM	0,2,0,8
RES	20
CMFEM	0,2,\$+1
DATA	08,0,1034,0
CCRFM	0,21,\$+1
CMFEM	0,2,0,8
RES	20
CMFEM	0,2,\$+1
DATA	08,0,990,0
CCRFM	0,21,\$+1
CMFEM	0,2,0,8
RES	20
CMFEM	0,2,\$+1
DATA	08,0,946,0
CCRFM	0,21,\$+1
CMFEM	0,2,0,8
RES	20
CMFEM	0,2,\$+1
DATA	08,0,902,0
CCRFM	0,21,\$+1
CMFEM	0,2,0,8
RES	20
CMFEM	0,2,\$+1



DATA	0	8	0,	858,	0
CCRD	0	0,	21,	\$+1	
CWFM	0	0,	2,	0,	8
MWFM	2	0,	2,	\$+1	
\$STARIP	0	0,	2,	\$+1	
CWFM	0	8	0,	814,	0
DATA	0	0,	21,	\$+1	
CCRD	0	0,	2,	0,	8
CWFM	0	0,	2,	\$+1	
MWFM	2	0,	2,	NUMS	
\$NUMMER1	0	8	0,	422,	0
CWFM	0	0,	22,	\$+1	
DATA	0	0,	2,	0,	4
CCRD	0	0,	6,	32,	32
CWFM	0	32,	32,	32,	32
MWFM	32,	32,	061,	32	
CHAR	32,	32,	32,	32	
CHAR	32,	32,	32,	32	
CHAR	0	62,	32,	32,	32
CHAR	32,	32,	32,	32	
CHAR	32,	32,	063,	32	
CHAR	32,	32,	32,	32	
CHAR	32,	32,	32,	32	
CHAR	0	64,	32,	32,	32
CHAR	32,	32,	32,	32	
CHAR	32,	32,	064,	32	
CHAR	32,	32,	32,	32	
CHAR	32,	32,	32,	32	
CHAR	0	66,	32,	32,	32
CHAR	32,	32,	32,	32	
CHAR	32,	32,	067,	32	
CHAR	32,	32,	32,	32	
CHAR	32,	32,	32,	32	
CHAR	0	70,	32,	32,	32
CWFM	0	0,	2,	\$+1	
DATA	0	8	0,	378,	0
CCRD	0	0,	22,	\$+1	
CWFM	0	0,	2,	0,	4
MWFM	0	61,	062,	063,	064
CHAR	0	65,	066,	067,	070
CHAR	0	71,	060,	061,	062
CHAR	0	63,	064,	065,	066
CHAR	0	67,	070,	071,	060
CHAR	0	61,	062,	063,	064



\$SBRAD1	RES	0,2,0,8
\$SBRAD2	RES	3
	DATA	0,2,\$+1
	CCRD	88,0,1739,0
	CMEM	0,21,\$+1
	CMEM	0,2,0,8
\$SBRAD3	RES	3
\$SBRAD4	RES	9
	DATA	0,2,\$+1
	CCRD	88,0,1694,0
	CMEM	0,21,\$+1
	CMEM	0,2,0,8
\$SBRAD5	RES	3
\$SBRAD6	RES	9
	DATA	0,2,\$+1
	CCRD	88,0,1650,0
	CMEM	0,21,\$+1
	CMEM	0,2,0,8
\$SBRAD7	RES	12
	DATA	0,2,\$+1
	CCRD	88,0,1606,0
	CMEM	0,21,\$+1
	CMEM	0,2,0,8
\$SBRAD8	RES	12
	DATA	0,2,\$+1
	CCRD	88,0,1562,0
	CMEM	0,21,\$+1
	CMEM	0,2,0,8
\$SBRAD9	RES	19
	DATA	0,2,\$+1
	CCRD	88,0,1519,0
	CMEM	0,21,\$+1
	CMEM	0,2,0,8
\$SBRAD10	RES	19

CHFM	0,2,\$+1
DATA	0
CCRC	RR,0,1474,0
CHFM	0,21,\$+1
CHFM	0,2,0,8
RES	1
\$SRADI1 REC	1
CHFM	0,2,\$+1
DATA	0
CCRC	RR,0,1430,0
CHFM	0,21,\$+1
CHFM	0,2,0,8
RES	2
\$SRADI2 REC	6
\$SBBADI3 REC	12
CHFM	0,2,\$+1
DATA	0
CCRC	RR,0,1386,0
CHFM	0,21,\$+1
CHFM	0,2,0,8
RES	2
\$SRADI4 REC	6
\$SBBADI5 REC	12
CHFM	0,2,\$+1
DATA	0
CCRC	RR,0,1342,0
CHFM	0,21,\$+1
CHFM	0,2,0,8
RES	2
\$SRADI6 REC	6
\$SBBADI7 REC	12
CHFM	0,2,\$+1
DATA	0
CCRC	RR,0,1298,0
CHFM	0,21,\$+1
CHFM	0,2,0,8
RES	20
CHFM	0,2,\$+1
DATA	0
CCRC	RR,0,1254,0
CHFM	0,21,\$+1
CHFM	0,2,0,8
RES	20
CHFM	0,2,\$+1
DATA	0
CCRC	RR,0,1210,0
CHFM	0,21,\$+1
CHFM	0,2,0,8

RES	20	0,2,\$+1
CWFM	0	
DATA	88	0,1166,0
CCRD	0	21,\$+1
CWFM	0	2,0,8
RES	20	0,2,\$+1
CWFM	0	
DATA	88	0,1122,0
CCRD	0	21,\$+1
CWFM	0	2,0,8
RES	20	0,2,\$+1
CWFM	0	
DATA	88	0,1078,0
CCRD	0	21,\$+1
CWFM	0	2,0,8
RES	20	0,2,\$+1
CWFM	0	
DATA	88	0,1034,0
CCRD	0	21,\$+1
CWFM	0	2,0,8
RES	20	0,2,\$+1
CWFM	0	
DATA	88	0,990,0
CCRD	0	21,\$+1
CWFM	0	2,0,8
RES	20	0,2,\$+1
CWFM	0	
DATA	88	0,946,0
CCRD	0	21,\$+1
CWFM	0	2,0,8
RES	20	0,2,\$+1
CWFM	0	
DATA	88	0,902,0
CCRD	0	21,\$+1
CWFM	0	2,0,8
RES	20	0,2,\$+1
CWFM	0	
DATA	88	0,858,0
CCRD	0	21,\$+1
CWFM	0	2,0,8









[illegible]

CCCC \*  
FORM 0240-1  
24,0

```

0000 060,00
0001 061,01
0002 062,02
0003 063,03
0004 064,04
0005 065,05
0006 066,06
0007 067,07
0008 070,010
0009 071,011
0010 046,012
0011 075,013
0012 047,014
0013 072,015
0014 076,016
0015 046,017
0016 053,020
0017 01,021
0018 02,022
0019 03,023
0020 04,024
0021 05,025
0022 06,026
0023 07,027
0024 010,030
0025 011,031
0026 077,032
0027 056,033
0028 051,034

```

LH BRACKET  
 .LT. EOL  
 - J K  
 M L  
 N C  
 P  
 R  
 CARRIAGE RET  
 \$  
 RH \*  
 BRACKET  
 SEMIC  
 BLANK  
 /  
 S  
 T  
 " V  
 QWWWWWWW  
 X Y  
 Z  
 EOL  
 ,  
 (

NUMBER SIGN  
BACKWARDS SLANT  
QUESTION MARK

ADDRESS	DISPLAY	FOR	PIN	OF	L . P.	STRIKE
ADDRESS	DISPLAY	FOR	PIN	OF	L . P.	STRIKE
ADDRESS	DISPLAY	FOR	PIN	OF	L . P.	STRIKE
ADDRESS	DISPLAY	FOR	PIN	OF	L . P.	STRIKE
ADDRESS	DISPLAY	FOR	POT	OF	KEY/FUN	WORD
ADDRESS	DISPLAY	FOR	POT	OF	DATA	WORD
ENABLE	DISPLAY	L . P.	INTERRUPTS			
ENABLE	DISPLAY	L . P.	INTERRUPT			
ENABLE	DISPLAY	KEYBOARD				

\$EOMPEN	EOM	031073
FCM		031067
\$EOMPIN	EOM	031057
FCM		031037
\$EOMPCT	EOM	031076
FCM		031075
\$ENRPN	EOM	031110
FCM		031120
\$ENRKEY	EOM	030235



[illegible]

# INITIAL DISTRIBUTION LIST

	No. Copies
Defense Documentation Center Cameron Station Alexandria, Virginia 22314	20
Library Naval Postgraduate School Monterey, California 93940	2
Naval Ship Systems Command Code 2052 Department of the Navy Washington, D. C. 20360	1
Prof. Mitchell Cotton Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	3
Lt. Andrew T. Dietzler 306 St. Nicholas Midland, Michigan 48640	2
Mr. Robert L. Limes Department of Electrical Engineering Computer Facility Naval Postgraduate School Monterey, California 93940	1



## DOCUMENT CONTROL DATA - R &amp; D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Naval Postgraduate School Monterey, California 93940		2a. REPORT SECURITY CLASSIFICATION Unclassified	
		2b. GROUP	
3. REPORT TITLE TIME SHARING TASK CONTROL FOR A HYBRID COMPUTER SIMULATION LABORATORY			
4. DESCRIPTIVE NOTES (Type of report and, inclusive dates) Master's Thesis (April 1967)			
5. AUTHOR(S) (First name, middle initial, last name) Andrew John Dietzler			
6. REPORT DATE April 1969		7a. TOTAL NO. OF PAGES 172	7b. NO. OF REFS 12
8a. CONTRACT OR GRANT NO.		9a. ORIGINATOR'S REPORT NUMBER(S)	
b. PROJECT NO.			
c.		9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
d.			
10. DISTRIBUTION STATEMENT Distribution of this Document is unlimited.			
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY Naval Postgraduate School, Monterey, California 93940	
13. ABSTRACT  The study of time sharing system parameters and design is undertaken. On-line and hybrid simulation programmer's demands for interactive digital computing time are time inefficient for modern high speed computers, hence the motivation for time shared computing systems. The techniques for achieving time sharing are studied, then applied to the problems of a real time, on-line hybrid simulation and batch processing system. Subroutines required for implementation of a task oriented time sharing capability are put forward with specific proposals for use. System improvements to accomplish the goals of a general time sharing system are introduced and discussed.			

14

### KEY WORDS

LINK A

LINK B

LINK C

ROLE

WT

ROLE

WT

ROLE

WT

- (1) TIME-SHARING COMPUTERS
- (2) ON-LINE DISPLAYS
- (3) HYBRID SYSTEMS



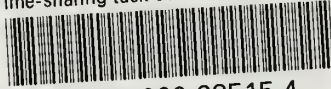






thesD572

Time-sharing task control for a hybrid c



3 2768 000 98515 4

DUDLEY KNOX LIBRARY